CIRCUITS AND ARCHITECTURE FOR BIO-INSPIRED AI ACCELERATORS

by
Gaspar Tognetti

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland
October 2020

Abstract

Technological advances in microelectronics envisioned through Moore's law have led to powerful processors that can handle complex and computationally intensive tasks. Nonetheless, these advancements through technology scaling have come at an unfavorable cost of significantly larger power consumption, which has posed challenges for data processing centers and computers at scale. Moreover, with the emergence of mobile computing platforms constrained by power and bandwidth for distributed computing, the necessity for more energy-efficient scalable local processing has become more significant. Unconventional Compute-in-Memory architectures such as the analog winner-takes-all associative-memory and the Charge-Injection Device processor have been proposed as alternatives.

Unconventional charge-based computation has been employed for neural network accelerators in the past, where impressive energy efficiency per operation has been attained in 1-bit vector-vector multiplications, and in recent work, multi-bit vector-vector multiplications. In the latter, computation was carried out by counting quanta of charge at the thermal noise limit, using packets of about 1000 electrons. These systems are neither analog nor digital in the traditional sense but employ mixed-signal circuits to count the packets of charge and hence we call them Quasi-Digital. By amortizing the energy costs of the mixed-signal encoding/decoding over compute-vectors with many elements, high energy efficiencies can be achieved.

In this dissertation, I present a design framework for AI accelerators using scalable compute-in-memory architectures. On the device level, two primitive elements are

designed and characterized as target computational technologies: (i) a multilevel non-volatile cell and (ii) a pseudo Dynamic Random-Access Memory (pseudo-DRAM) bit-cell. At the level of circuit description, compute-in-memory crossbars and mixed-signal circuits were designed, allowing seamless connectivity to digital controllers. At the level of data representation, both binary and stochastic-unary coding are used to compute Vector-Vector Multiplications (VMMs) at the array level. Finally, on the architectural level, two AI accelerator for data-center processing and edge computing are discussed. Both designs are scalable multi-core Systems-on-Chip (SoCs), where vector-processor arrays are tiled on a 2-layer Network-on-Chip (NoC), enabling neighbor communication and flexible compute vs. memory trade-off. General purpose Arm/RISCV co-processors provide adequate bootstrapping and system-housekeeping and a high-speed interface fabric facilitates Input/Output to main memory.

Thesis Readers

Dr. Andreas G. Andreou (Primary Advisor)
Professor
Department of Electrical and Computer Engineering
Johns Hopkins University

Dr. Ralph E. Cummings
Professor
Department of Electrical and Computer Engineering
Johns Hopkins University

Dr. Philippe O. Pouliquen
Professor
Department of Electrical and Computer Engineering
Johns Hopkins University

This thesis is dedicated to my loving wife, Rosie, who has supported me in every sense, every step of the way.

Acknowledgments

I would like to thank my advisor, Andreas G. Andreou for accepting me into his PhD program. He has providing endless stimulating discussions, feedback, and insight into many crazy ideas, many of which are shown in this dissertation. We worked hard and we also played hard.

I would like to thank Ralph Etienne-Cummings and Philippe O. Pouliquen for agreeing to participate as members of my dissertation committee and for providing great discussions about work and life, whether if it was in the lab in Barton Hall, mountain biking, or rock climbing.

I would like to thank my lab mates for providing invaluable collaborations and discussions regarding my work. I believe that peer collaborations are key to innovative ideas and therefore are the most valuable in the course of a PhD career. For this, I would like to thank Martin Villemur, Kayode Sanni, Jonah Sengupta, Tomas Figliolia, Christos Sapsanis, Dan Mendat, Alejandro Pasciaroni, Kate Fischl, Francisco Tejada, and Pedro Julian.

I would like to thank the folks at Northrop Grumman Corporation, specifically, Nishant Zachariah, Sergio Montano, Isidorox Doxas, Kayode Sanni, Quentin Swain, John Goldsmith, Bingxin Tian, Banesh Chandrasekhara, Alex Toulouse, Louise Sengupta, Tom Petty and Doyle Nichols, who have provided the highest quality of management and engineering support during my last year.

I would like to thank my parents, Pablo Tognetti and Gloria Tognetti, for opening

the doors that led me to this point.

I would like to thank my siblings, Lila, Coni, and Celia, for their support and love during these years.

I am grateful for my family in Montana, for helping and encouraging me every step of the way. Tom and Laurie Bartoletti have been my second parents, and Maria, Louis, Juliana, Jade and Jack have been my second siblings.

Lastly, I am thankful for my close friends, Nacho & Juan, Juan & Garo & Adrian, Mario & Mario & Enzo, Tom & Reina, and Shannon, who have given me incredible friendship and support during my years at Johns Hopkins.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

To meet the scientific demand for future data-intensive processing for every day mundane tasks such as searching via images to the uttermost serious health care disease diagnosis in personalized medicine, we urgently need a new cloud computing paradigm and energy efficient i.e. "green" technologies. In some ways, the human brain is the ultimate machine for BIGDATA processing. We all have an uncanny ability to remember and recall facts (*big volume*), that are stored and recalled as a result of our daily multi-sensory experience with the world (*big variety*), and are derived from signals that inundate our senses every single moment (*big velocity*). Everything is done in real-time in a resilient and robust multiprocessor architecture that is so energy efficient that it barely produces enough heat to keep itself warm in the winter. Hence, we believe that a brain-inspired approach that employs unconventional processing offers an alternative paradigm for BIGDATA computing. In a neuromorphic information-processing system-architecture, one desires both pattern processing in space and time to extract features in symbolic sub-spaces as well as natural language processing to provide contextual and semantic information in the form of priors.

In the early 1990s, Carver Mead exposed the potential of neuromorphic electronic systems [1] and showed that the scaling of standard digital circuits was limited by physical factors [2]. Neuromorphic circuits have shown to be capable of solving problems whose complexity grow exponentially with the size of the input [3, 4]. This is

especially the case in neural networks which have combinatorial interconnectivity, and are ubiquitously used to solve inference problems collectively by the network per se.

The first silicon neural engines were analog and inspired by the similarities between biological neurons and the physical properties of semiconductors. Although attractive and elegant, the advent Moore's law [5] has allowed for digital Von Neumann architectures to scale into deep sub-micron processes; processing power has also scaled and has dwarfed neuromorphic computing for decades. Scaling computers with Moore's law has provided a low-cost solution to increasingly complex problems over the years. Although reaching incredibly small feature sizes, such as the projected TSMC 3nm process in 2021 [6], contemporary processors are encountering the physical limitations predicted by Mead [2]. In order to address these limitations, computer architecture needs to undergo a fundamental change in computing paradigm; unconventional compute-in-memory architectures such as the analog winner-takes-all associative-memory, charge-injection device processor, and analog-array processing have been proposed as alternatives.

Charge-based circuits were used for vector quantization in the early 1990s [7], where charge-based analog circuits performed Euclidean distance computation and search, something that is also done today in some AI accelerators. Subsequently it was recognized that charge domain processing can be employed in Application-Specific Integrated Circuits (ASICs) in computations ranging from simple weighted multiplication-and-addition [8–10] and more recently in high-frequency analog Fast Fourier Transforms (FFTs) [11]. Charge-based computation has also been employed in neural-network accelerators in the past [12, 13], where impressive energy efficiency per-operation has been attained when performing 1-bit bitwise Vector-Vector Multiplication (VMM) [14, 15], and in recent work, also multi-bit vector-vector multiplication [16]. While not targeted specifically for Artificial Intelligence (AI), charge-based neuromorphic circuits by Noack [17] and more recently by Mayr & Noak [18, 19] perform

mixed-signal low precision dot-product operations in arrays of "neuronal" elements. Subsequently, a charge-based, passive vector dot-product multiplier circuit followed by a Successive-Approximation Register (SAR) Analog-to-Digital Converter (ADC) was also reported in the literature [20]. A six-transistor, one-capacitor (6T+1C) SRAM based binary compute-in-memory accelerator also computes in the charge domain [21]. A similar approach was used in earlier work where a pseudo-DRAM is employed to store the binary state vectors and computations are done using 1-bit [15] or multi-bit [16] inputs in the charge domain.

A number of charge-based architectures for low precision Vector-Vector Multiply-and-Accumulate (VV MAC) compute-in-memory units have also been investigated by Sanni [22, 23]. In the latter, computation is carried out by counting quanta of charge at the thermal noise limit, using packets of about 1000 electrons. These systems are neither analog nor digital in the traditional sense but employ mixed-signal circuits to count the packets of charge and hence we call them Quasi-Digital. By amortizing the energy costs of mixed-signal encoding/decoding over a large number of elements in compute-vectors, high energy efficiencies can be achieved.

In this dissertation, I present a design framework for AI accelerators using scalable compute-in-memory architectures. Chapter 2 introduces the fundamental limitations of Von Neumann architectures and sheds light on Compute-in-Memory (CiM) crossbars as an elegant solution to the problem. A brief description of the binary and unary compute-modalities utilized in the thesis are presented as well as a overview on how to configure crossbars in both modalities. Lastly, a bibliographical reference showing two state-of-the-art CiM processors is shown: (i) IBM's TrueNorth, and (ii) Intel's Loihi. The two subsequent chapters provide device and circuit-level of descriptions of the hardware used in ASICs while targeting two technologies: (i) Non-Volatile Memory (NVM) and (ii) pseudo Dynamic Random-Access Memory (pseudo-DRAM). Chapter 3 explores NVM multi-level cells for compute-in-memory processing, where two ASICs

were designed: (i) an analog test-chip fabricated in CMOS 55nm used to create both analog and digital models of NVM arrays, allowing simulation and emulation of larger systems in reconfigurable hardware platforms (FPGAs), and (ii) a mixed-signal compute-in-memory test-chip fabricated in the same technology. Chapter 4 delves into pseudo-DRAM primitives and explores their use in CiM architectures. At the array and device level of description, a mixed-signal compute-in-memory ASIC was designed and fabricated in CMOS 55nm. Chapters 5 & 6 describe a multi-core SoC architecture for edge-computing AI applications which share the following features. Both designs include a 2-layer Network-on-Chip (NoC), providing a 2D mesh-grid topology where mixed-signal cores are instantiated at each node. Cores can communicate with each other as well as with conventional ARM/RISCV co-processors, which are used for bootstrapping and housekeeping. A token-ring network allows data packets communicate with a high-speed, high-bandwidth I/O interface on a row-basis. Chapter 5 utilizes NVM crossbar arrays in the CiM cores, for which custom analog and mixed-signal circuits were designed. Chapter 6 utilizes results from previously designed ASICs (described in Chapter 4), and leverages pseudo-DRAM primitives for performing VMM operations. The latter is the culmination of this thesis and comprises a bundle of novel digital and mixed-signal circuits designed to minimize energy consumption and maximize system throughput. Finally, Chapter 7 shows conclusions, provides a discussion, and point out future work.

Chapter 2

Background & Preliminary Concepts

The shortcomings of conventional approaches have forced computer scientists and engineers to borrow paradigms from biology to solve problems in sensory perception and machine intelligence. In addition to handling noisy and even novel inputs, neuromorphic systems have two other desirable features: fault tolerance and massive parallelism and one of their salient features is the co-location of processing and memory in what it is often referred to as "associative processing" [24]. Hardware AI inference has looked in the past towards associative processing to circumvent the limitations of Von Neumann architectures [25]. The work by Lee et al. [26, 27] and Herrmann [28] emphasized digital VLSI content addressable and associative memories for specialized computing engines. The term Processor-in-Memory (PIM) was coined by Gokhale [29] and it was argued that intelligent RAM could be could be quite effective parallel processing [30]. The trade-offs in PIM were investigated by [31]. However, no digital implementation of an associative processing system can capture the central idea of a physical system that is able to store and process information like the brain. Neural paradigms for associative memories using mixed-signal circuits have been proposed and investigated in the past [14, 32–37]. The computational capability of these models has been demonstrated with problems in pattern recognition, vector-quantization,

novelty filtering, and optimization.

In this chapter, the fundamental bottleneck of conventional computing, i.e. the Von Neumann bottleneck will be exposed. Next, a neuromorphic solution to a subset of problems is presented: Compute-in-Memory (CiM). The former enables the use of dedicated, non-conventional compute structures to serve a building blocks of larger processors. Later, different compute modalities will be discussed using the same underlying structure, and finally, a literature review of some of the state-of-the-art architectures is presented.

2.1 Compute-in-Memory (CiM) Paradigm

The concept of the Compute-in-Memory stems from the high energy expenditure that comes from shuttling data between memory and arithmetic-logic-units (ALUs), or

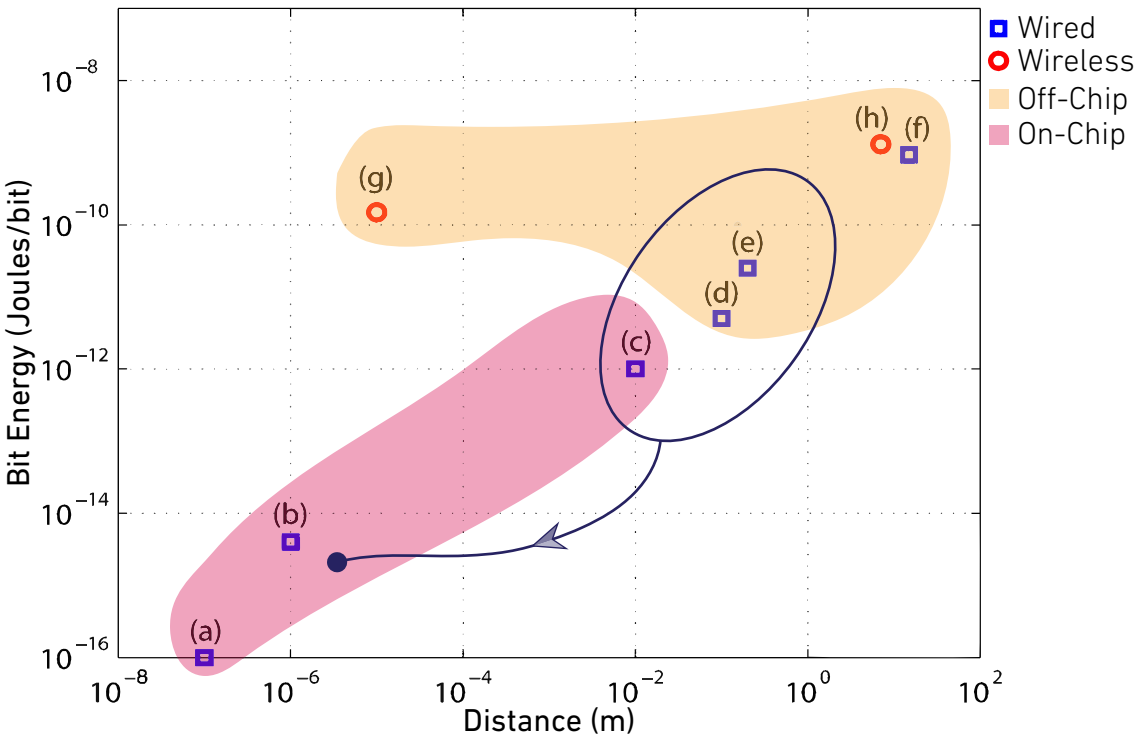


Figure 2.1 *Energy per bit as a function of distance in a CMOS chip. Figure Adapted from Marwick & Andreou [38]. (a–c) Switching of a metal line across a die, (d) electrical chip-to-chip link, (e) optical chip-to-chip link, (f) Firewire link, (g) wireless near-field communication between chips, and (h) Ultra-Wide-Band radio link.*

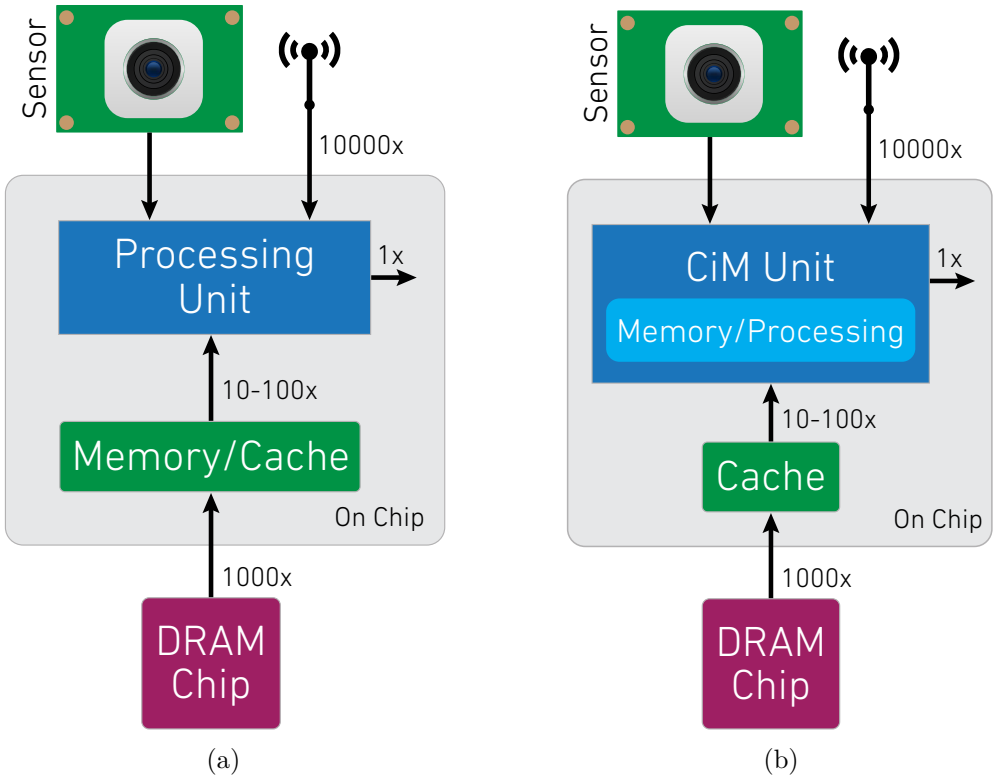


Figure 2.2 ***(a)*** *Standard computing Vs.* ***(b)*** *Compute-in-Memory paradigm.*

Processing-Units (PUs). Figure 2.1 shows a plot adapted from Marwick & Andreou [38], where different bit-flip energies were computed while considering traveled distance of information. Intra-chip switching (0-2.5v) energy increases four orders of magnitude, starting from a short wire (∼femto Joules) all the way to a 1cm wire across a die. Moreover, an additional two to three orders of magnitude of energy increase are seen when data travels off-chip, either by a wired or a wireless link.

The plot from Fig. 2.1 has been abstracted into an illustrative computing architecture, shown in Fig. 2.2. From the figure, the processing-unit is assumed to be connected to a set of peripherals, by means of which data is brought from either local memory, or an off-chip communication link. For example, video imagery could be captured with a camera (sensor), sent to the processing-unit, who later performs a De-Bayer filter on RGB pixels. The processing-unit is assumed to have an output that burns 1× quantum of energy. If the parameters for the De-Bayer filter are stored

off-chip or in the cloud, a 1000×–10000× of energy/bit will be spent in fetching these parameters. In this scenario, the processing-unit's compute energy will be negligible to that of fetching filter coefficients.

On-chip Memory/Cache could store frequently accessed coefficients, thus allowing a reduction of energy by a factor 1000×. Moreover, bringing the frequently accessed parameters closer to the processing-unit may be taken to the extreme, which entails embedding the processing and the memory into a monolithic structure. This is illustrated in Fig. 2.2b, where Memory has been included in the processing-unit. Under these assumptions, entries (c–e) in Fig. 2.1 would move down to the sector next to entry (b), indicated with a dark blue arrow.

2.1.1 Analog Vs. Digital Computation

In-memory computing may take place either in the analog or the digital domain. However, since memories are naturally arranged in crossbars, it results natural to attempt using common nodes across rows/columns to perform some sort of computation in the analog domain. In the context of this thesis, the former could take place either by summing currents on a common node, or by averaging charge on a common plate capacitance. If the output of the analog computation is to be used in a digital system, then it must be converter appropriately using an Analog-to-Digital Converter (ADC or A/D). Similarly, if the inputs to the memory block originate in the digital domain, then they must be either encoded properly, or converted to analog (DAC or D/A) prior to computing. The former description is captured by Fig. 2.3a, where both DAC and ADC are incorporated to form a digital-input, analog-compute, digital-output system.

On the other hand, the CiM block may be capable of performing the computation directly in the digital domain. This could be accomplished by hard-wiring digital data from a memory row/column and embedding adders in the memory cells. The

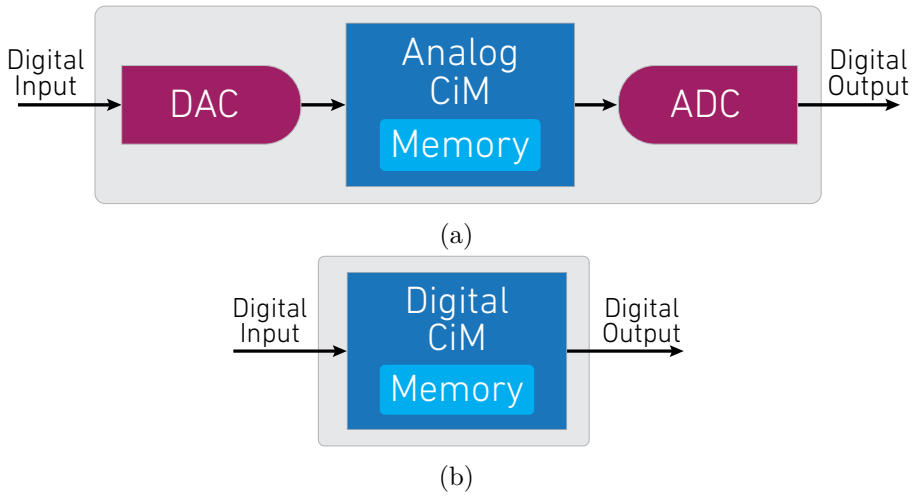


Figure 2.3 ***(a)*** *Analog vs.* ***(b)*** *digital Compute-in-Memory.*

architecture of the memory changes, yet the computation is carried out locally and with no external effort. Figure 2.3b shows such a system, which does not require DACs or ADCs.

In this thesis, both Non-Volatile Memory (NVM) and DRAM technologies are addressed and evaluated as analog and mixed-signal computational macros.

2.2 Compute Architecture & Arithmetic

Figure 2.4a shows a 2D memory crossbar which is utilized via random-access; an address ($Addr$) is used to select a row which is to be read or written, determined by the RD and EN signals. Row access consists of a decoder, and column access is comprised of sense-amplifiers that are used to quickly sense data that is stored in memory for subsequent recall. This architecture lends itself for adding (or reusing) common nets for broadcasting data across the array and performing a computation against stored bits. Figure 2.4b shows such a device, where added horizontal wires are

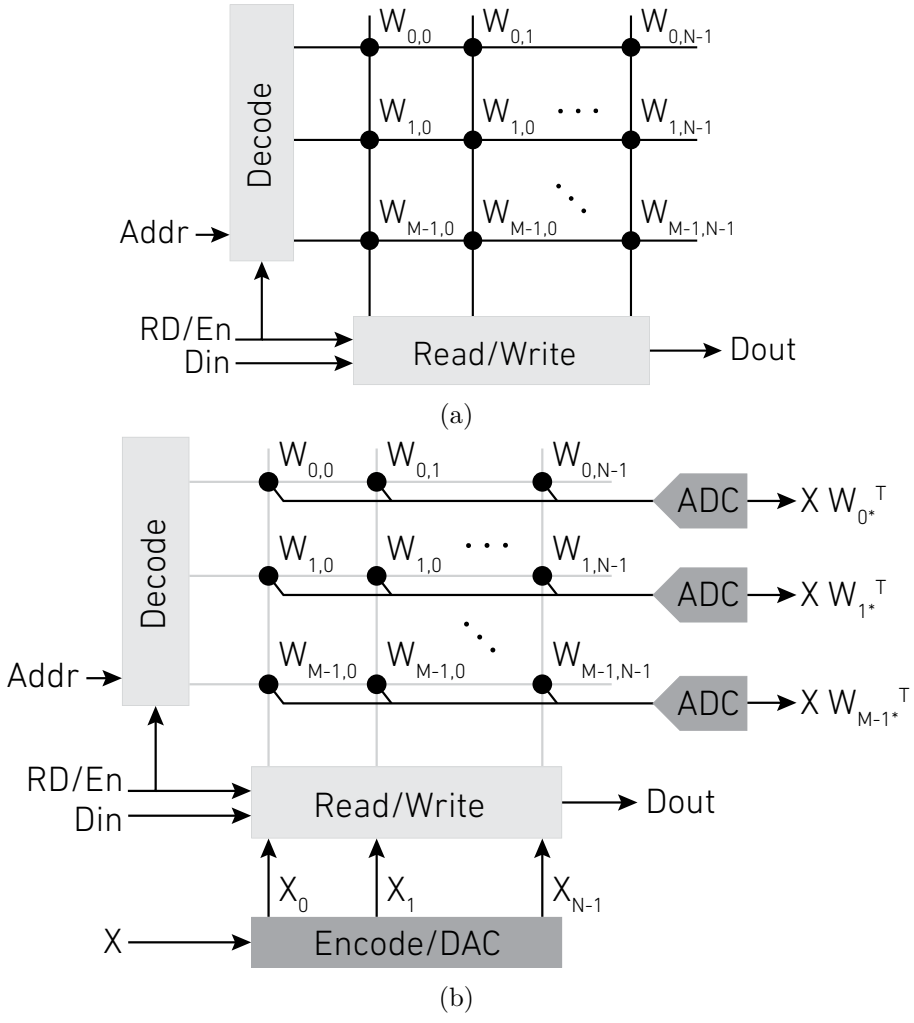


Figure 2.4 ***(a)*** *Random-Access-Memory crossbar and* ***(b)*** *memory crossbar used for computing.*

used to read-out analog signals (voltage or current), which in turn are the result of an element-wise computation between an input, $\mathbf{x}$, and parameters $\mathbf{w^T_{m*}}$. In this case, the dots in the figure represent a 1-bit AND operation between stored data and an input.

2.2.1 Vector-Matrix Multiplication (VMM)

When the outputs of all rows in Fig. 2.4b are considered, the operation consists of a Vector-Matrix Multiplication (VMM), yielding the following expression:

$$\mathbf{z} = \mathbf{x}\,[\mathbf{w^T_{0*}}, \mathbf{w^T_{0*}}, \ldots, \mathbf{w^T_{M-1*}}] \tag{2.1}$$

$$\mathbf{z} = \mathbf{x}\,W^T. \tag{2.2}$$

Vector-matrix multiplications are ubiquitous computations carried out in Deep Learning [39], which make them a clear candidate to optimize when it comes to non-conventional compute paradigms. In Deep Learning, a multi-feature extraction process consists of convolving (matching) an input dataset against several features (patterns). Von Neumann architectures are especially inefficient at this task, since they require that all operands be fetched from memory prior to a computation. It has been shown that in these scenarios, orders of magnitude of energy savings may be achieved (compared to Von Neumann) when one of the operands (the one that is not required to change) is stored at the same location as the arithmetic unit. Moreover, the former can be tailored to a specific task, such as dot-products, thus increasing energy savings even further.

2.3 Compute Modality

Within the scope of this thesis, two compute modalities are explored: (i) binary and (ii) unary. The compute modality is coupled to data coding, where binary weighted representation can be used to represent 2^N discrete values with N-bits, and unary coding (e.g. PWM) can represent 2^N discrete values with 2^N bits. In this section, both coding schemes and their variants (when applicable) will be discussed, as well as VMM implementation details concerning the previously explained crossbar CiM architecture, thus concepts from this section will be used across this thesis

2.3.1 Binary Coding

Binary coding allows quantized data to be represented in the binary domain, where N-bits are required to represent 2^N discrete levels. Literature also describes binary coding as a $\log_2$ compression, where the base, 2, arises naturally due to the binary nature of digital computers. In a binary coded scheme an N-bit binary number, $\mathbf{x} = (x_{N-1}, x_{N-2}, \ldots, x_0)$, can be represented in base 10, by performing a linear combination of weighted digits, as follows (2.3).

$$\mathbf{x} = \sum_{i=0}^{N-1} 2^i \, x_i. \tag{2.3}$$

Using this coding scheme, a column in the crossbar can be used to represent an M-bit number (because the crossbar has M-rows), or any combination of separate numbers as long as their combined accuracy is less, or equal to M-bits.

Binary Multiplication

The following diagram illustrates an example of the naive unsigned binary multiplication algorithm of two 3-bit numbers: $\mathbf{x} = (x_2\, x_1\, x_0)$, and $\mathbf{w} = (w_2\, w_1\, w_0)$.

		x_2	x_1	x_0
		w_2	w_1	w_0
0	0	x_2w_0	x_1w_0	x_0w_0
0	x_2w_1	x_1w_1	x_0w_1	0
x_2w_2	x_1w_2	x_0w_2	0	0
p_4	p_3	p_2	p_1	p_0

Considering the binary nature of the data and the underlying hardware, this result may be computed by adding and shifting the partial products, $x_i w_j$, yielding the following equation:

$$\mathbf{x}\,\mathbf{w} = \sum_{i=0}^{2N-2=4} 2^i\, p_i, \tag{2.4}$$

where p_i is the sum of the partial products of the $i^{th} - column$. This operation requires $2N-2$ steps to sum N^2 products, where N is the bit-precision of the numbers. A graphical representation is shown in Fig. 2.5, where the "<<" symbols represents a left-shift in hardware. The former could be implemented in time (shift register) or in parallel combinatorial logic. Given the nature of the data stored in the CiM crossbar, a time-multiplexed scheme is considered

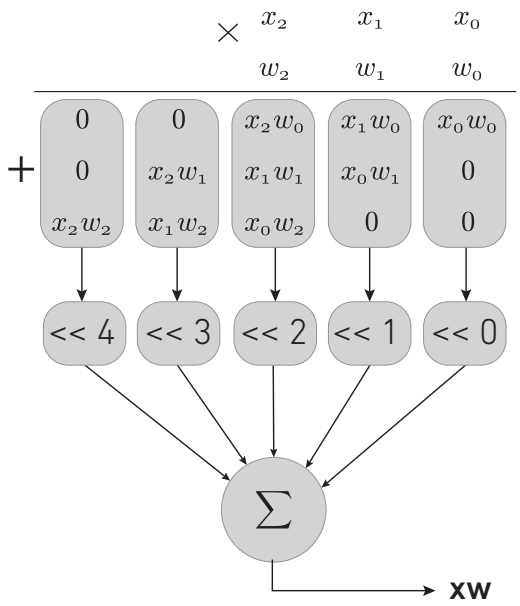


Figure 2.5 *Example of how to map a binary multiplication to hardware.*

Vector-Vector Multiplication (VVM)

Figure 2.6 shows the architecture for a CiM crossbar which can perform Vector-Vector Multiplications (VMMs). Inspired by Eq. 2.4, the crossbar may be configured to perform a VVM between stored M-bit parameters, W_i, and K-bit inputs, X_i. Parameters are stored in a binary coded fashion, where the position of a coefficient across rows represents the corresponding digit in the number. Most-Significant-Bits (MSBs) are stored towards the bottom array and Least-Significant-Bits (LSBs) are stored towards to top. The architecture performs the VMM operation in 2K-2 time steps, where at each step, k, the input vectors are shifted MSB first. The operation noted by a solid dot is a 1-bit multiplication and since the operation across rows

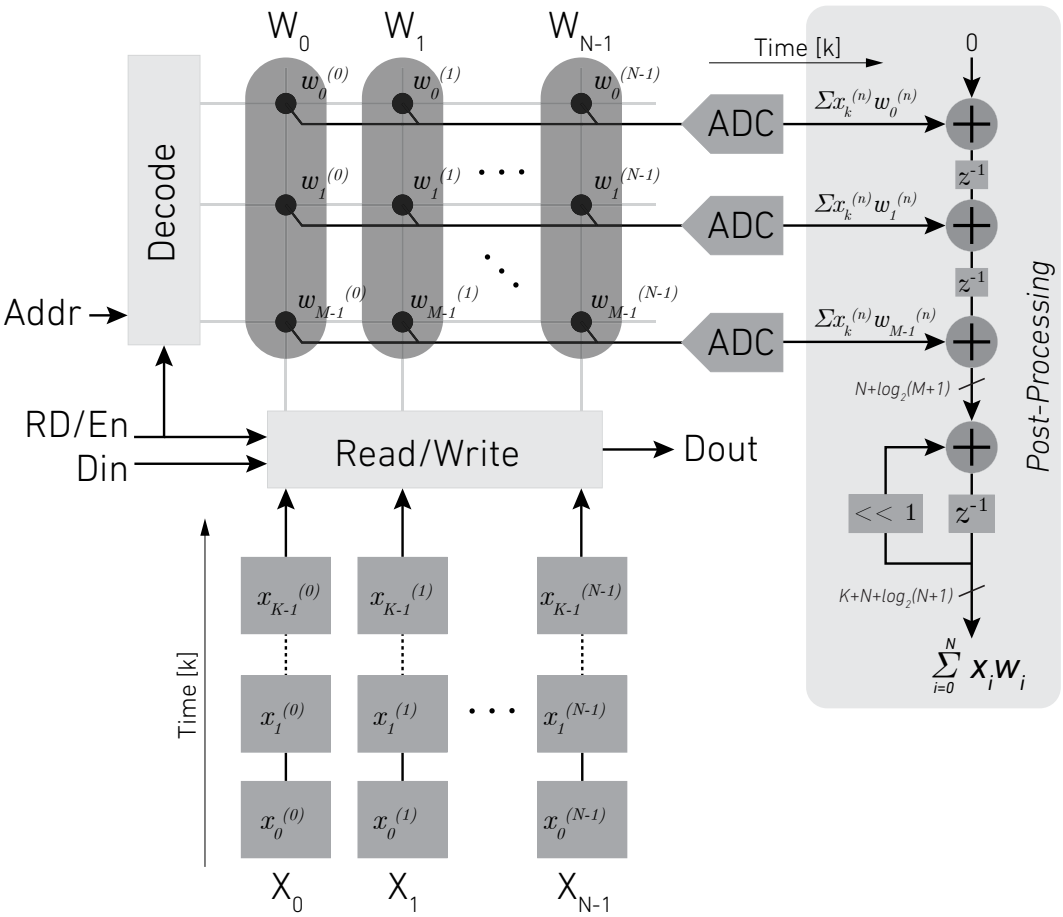


Figure 2.6 *Compute-in-Memory crossbar used for computing a Vector-Vector Multiplication (VVM) using binary coding.*

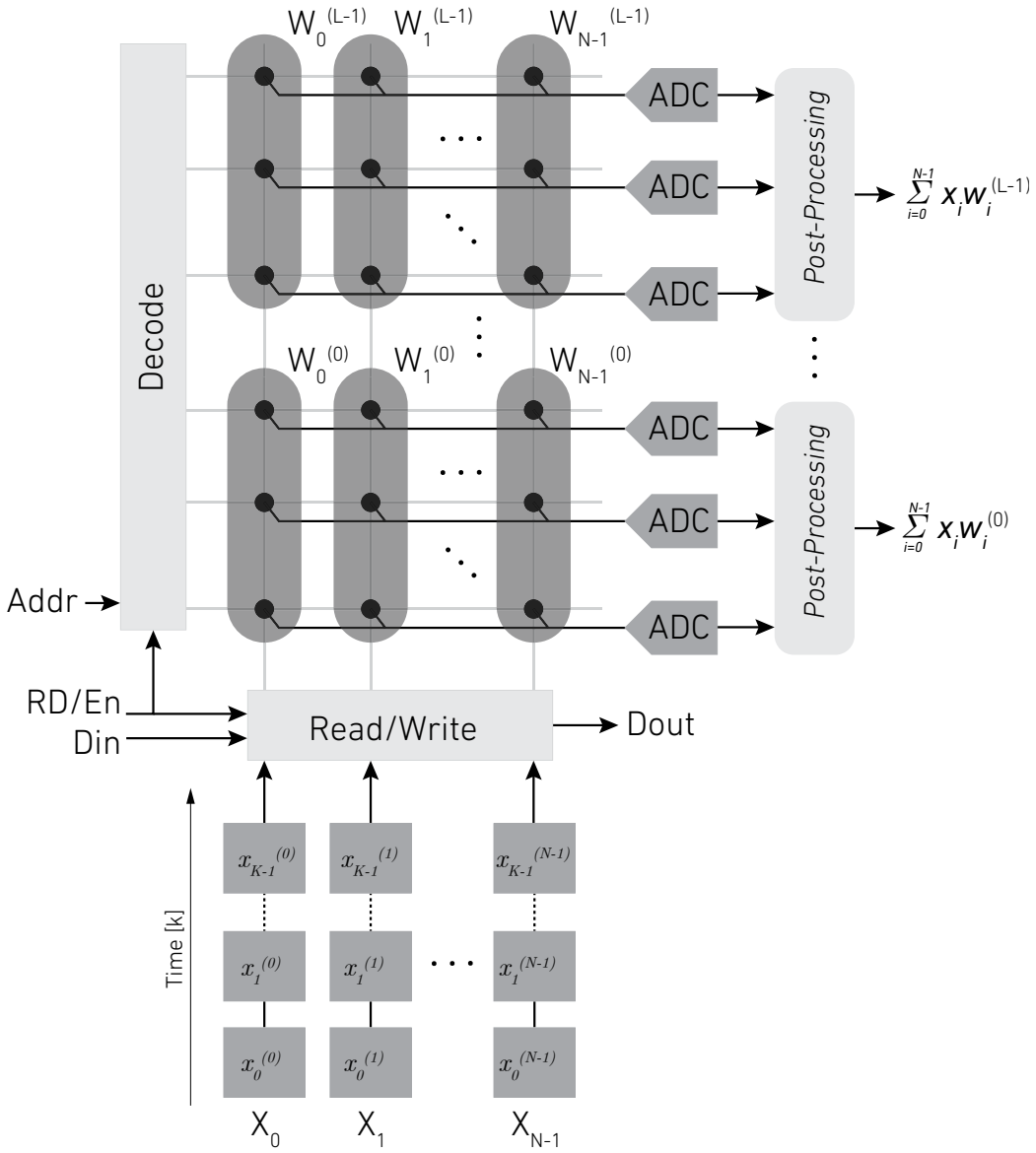


Figure 2.7 *Compute-in-Memory crossbar used for computing a Vector-Matrix Multiplication (VMM) using binary coding.*

converging in the ADC is linear, several columns are used to combine partial products. The gray area on the right of the figure shows the required digital post-processing hardware. At each time-step, k, the partial products are delayed and accumulated, resulting in a combined complete version of Eq. 2.4.

Vector-Matrix Multiplication (VVM)

The architecture shown in Fig. 2.6 can be expanded to compute a Vector-Matrix Multiplication (VMM) by utilizing more rows and isolating dedicated post-processing hardware. Such an architecture is shown in Fig. 2.7, where subsets of rows were combined to form vectors of different lengths, and subsequently multiplied by inputs. The right-hand side of the figure shows each vector coefficient as the inner product between weight vectors and input vectors.

It is important to notice that binary coding achieves the best possibly achievable spatial compression, but it is prone to errors; if no error-correction-coding (ECC) is used at the array level, a single bit flip could create a change in the number in a factor of one-half. This could be mitigated by incorporating redundancy in the binary data but escapes the scope of this thesis.

2.3.2 Unary Coding

The second type of coding considered in this thesis is *unary*; each bit of data represents a Least-Significant-Bit (LSB) of an N-bit binary number. Unary representation is also referred to as pulse-density-modulation, where the cumulative density of pulses in a binary vector represents the decimal value of the data. In the scope of pulse-density-modulation, two distinct encodings will be considered: (i) Pulse-Width-Modulation and (ii) Stochastic Modulation.

Pulse-Width-Modulation (PWM)

Pulse-Width-Modulation is achieved by comparing a constant to a ramp (counter), thus achieving a thermometer-type code. Figure 2.8a shows the memory crossbar where each parameter, w_i, populates an entire row with a spatial PWM. An all-ones input is subsequently applied to the array x times, spread out in a time-domain PWM. For each time-step, k, a post-processing block accumulates the results, thus achieving

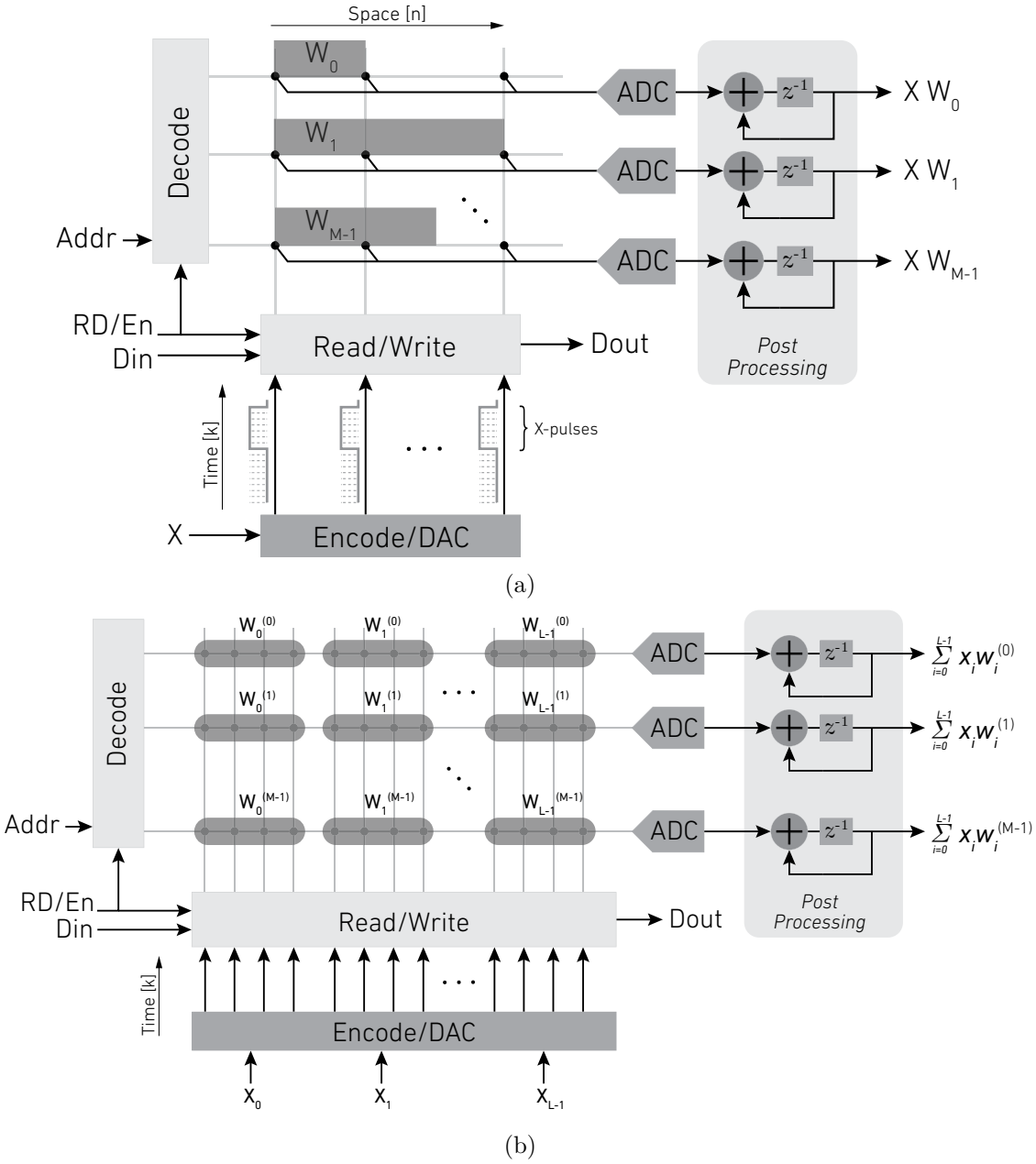


Figure 2.8 ***(a)*** *Compute-in-Memory crossbar used for computing a scalar-vector multiplication using PWM encoding.* ***(b)*** *Compute-in-Memory crossbar used for computing a Vector-Matrix Multiplication (VMM) using PWM encoding.*

the distributive product between x (scalar) and $\mathbf{w} = (w_{M-1}, w_{M-2}, \ldots, w_0)$ (vector), yielding $\mathbf{z} = x\,\mathbf{w}$ (vector). The precision of x can be arbitrarily large (at the expense of time), and the precision of w_i can be at most $\log_2(columns)$.

Decreasing the precision of w_i allows the array to harbor more parameters, thus

enabling the crossbar to compute Vector-Matrix-Multiplications. Such an architecture is shown in Fig. 2.8b, where the only difference is added complexity in the encoding scheme. The number of columns allocated for each parameter, w_j^i, determines its precision as $\log_2(columns)$. From the figure, each encoded input, x_j, is bundled and color-coded after passing through the encoding block.

Stochastic-Computing (SC)

Stochastic Computing (SC) is a low-cost alternative to conventional binary or PWM encoding. In the scope of unary encoding, SC is advantageous with respect to PWM in two main aspects: (i) an additional time-compression and (ii) a reduction in post-processing hardware may be achieved. Stochastic computing has existed since the 1960s, introduced by Gaines, Ribeiro, and Poppelbaum [40–42], and thoroughly reviewed by Alaghi & Hayes [43]. Stochastic computing relies on statistical properties of random-binary coding in order to simplify the hardware and architecture for arithmetic and logic units. Stochastic computing is also attractive due to its resilience to interference, where if a single bit is corrupted, only a one LSB (out of the full-range) will be affected, as opposed to a potential half of full-range in binary coding. In this section, the basics of SC will be reviewed, showing a brief example of a multiplication, encoding and decoding, and the generation of the random numbers. Later, it will be shown how the crossbar architecture discussed in this chapter will be adapted to compute Vector-Matrix-Multiplications (VMMs).

Multiplication

Let us consider two Bernoulli random variables, x and y, with associated probabilities p_x and p_y. Since both variables are independent, the distribution of their product, $z = x\,y$, will also be Bernoulli distributed with associated probability $p_z = p_x\,p_y$. In this way, random-binary-streams with individual Bernoulli samples (trials) can be used with a simple AND gate to perform a

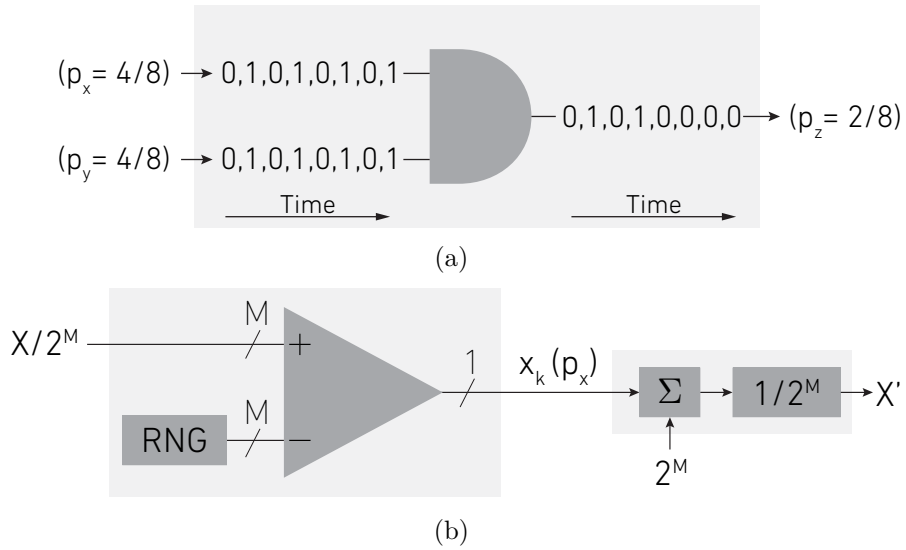


Figure 2.9 ***(a)*** *Example of a multiplication using stochastic computing (AND gate).* ***(b)*** *Encoding and decoding scheme in stochastic computing.*

multiplication, such as shown in Fig. 2.9a.

Encoding/Decoding

As shown on the left hand side of Fig. 2.9b, stochastic numbers can be generated with a random number generator (RNG) and a comparator. At each index k, the RNG creates a uniformly distributed number and compares it with an M-bit reference, X. If X greater than the random number (at index-k), then the output of the comparator will be high, otherwise the output will be low.

Each stochastic number, x_k, follows a Bernoulli distribution with a probability of 1, given as $p_x = X/2^M$. A block diagram of a stochastic decoder is depicted on the right hand side of Fig. 2.9b. From the figure, it can be seen that the normalized sum of x_k forms the decoded output X', which is a Binomially-distributed random variable. Assuming that the random numbers x_k are independent and identically distributed (i.i.d), X' has the following mean and variance:

$$\begin{aligned} \mu_x &= \frac{X}{2^M} \\ \sigma_x^2 &= \frac{p_x(1-p_x)}{2^M} \end{aligned} \tag{2.5}$$

where 2^M is the integration count. Following the central limit theorem, as $N = 2^M$ increases, X' tends towards a Normal distribution. Increasing the integration count results in a reduction of the standard deviation, making this variable suitable to bound the output error.

Random Number Generator: the LFSR

It has been shown in literature (Alaghi & Hayes and Gupta & Kumarsen [43]) that when a maximal length Linear-Feedback-Shift-Register (LFSR) is used to create a stochastic number, an integration count of 2^M is sufficient to achieve M-bits. Surprisingly, several techniques using LFSRs have been developed to create uncorrelated random-binary-streams for use in stochastic computing [44] and have proven to be more accurate than utilizing true random numbers. The reason lies behind the encoding; after a full period, an M-bit LFSR will generate all possible binary codes [0, $2^M - 1$] in a pseudo-random pattern, therefore comparing the LFSR contents to a constant will result in a deterministic number of pulses/values (encoded in time/space). Additionally, shifted codes generated from the same LFSR have uniform and bounded correlation [45], which makes them suitable for stochastic computing, and therefore will be used in this thesis.

Vector-Matrix Multiplication (VMM)

Figure 2.10 depicts a computational memory crossbar storing random-binary weights in a row-parallel fashion (space). When a random-binary pattern is applied to the input of the array, a scalar-vector multiplication is achieved between the input, x (scalar), and the stored weights, $\mathbf{w}$ (vector), resulting in the output vector $\mathbf{z} = x\,\mathbf{w}$. This architecture achieves the same result as that of the PWM-domain with a time saving of x-cycles, and a reduction in hardware of M-accumulators. This architecture is also suitable for column partitioning (as shown in Fig. 2.8b), where the array is capable of computing true vector-matrix multiplications. Each row may harbor

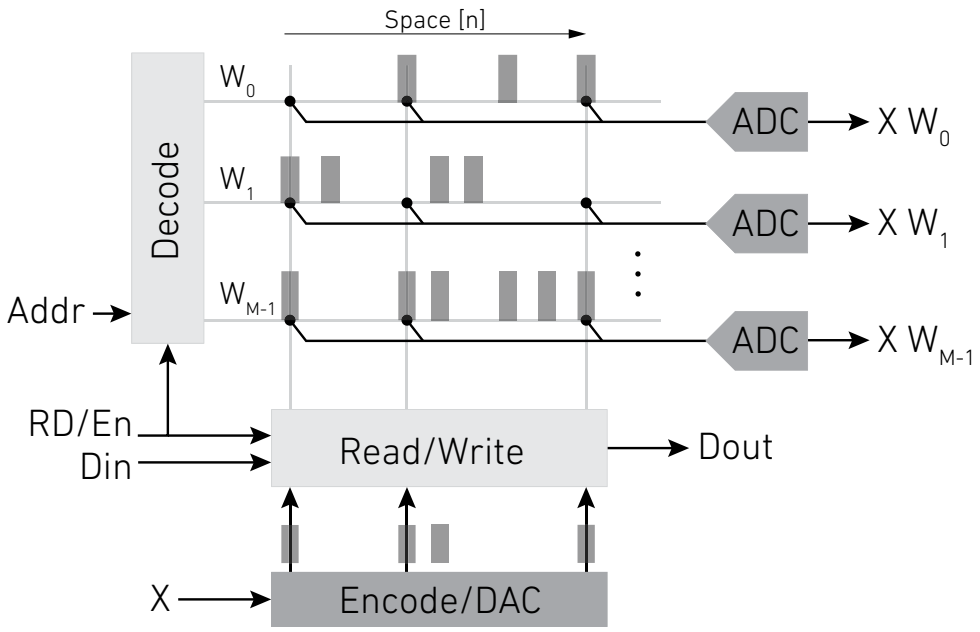


Figure 2.10 *Compute-in-Memory crossbar used for computing a Vector-Matrix Multiplication (VMM) using stochastic encoding.*

L coefficients, w_{il} in variable precision, as long as the cumulative precision of all parameters does not exceed that of $\log_2(columns)$. Similarly, the inputs, x_l, must be partitioned spatially into L slots.

2.4 Prior Work

TrueNorth

The state of the art today in Processing-in-Memory (PIM) technology today is the IBM TrueNorth (TN) Neurosynaptic System. TrueNorth was fabricated in CMOS 28nm and occupies a die area of approximately 4.3 cm^2, and burns 65mW during a typical object detection task [46] and classification thereof [47]. TrueNorth is a chip multi-processor [48, 49] with a tightly coupled processor/memory architecture, that results in energy efficient neurocomputing and it is a significant milestone to over 30 years of neuromorphic engineering. It comprises 4096 cores, each core with 65K of local memory (6T SRAM) -synapses- and 256 arithmetic logic units -neurons- that operate on a *unary* number representation and compute by counting to a maximum of 19 bits. The cores are event-driven using custom asynchronous and synchronous logic, and they are globally connected through an asynchronous packet switched mesh network on chip (NoC). The chip development board includes a Zyng Xilinx FPGA which does the housekeeping and provides support for standard communication support through an Ethernet UDP interface. The asynchronous Addressed Event Representation (AER) in the NoC is also exposed to the user for connection to AER based peripherals through a packet with bundled data full duplex interface. The unary data values represented on the system buses can take on a wide variety of spatial and temporal encoding schemes. Pulse density coding (the number of events N_e represents a number N), thermometer coding, time-slot encoding, and stochastic encoding are examples. Additional low-level interfaces are available for communicating directly with the TrueNorth chip to aid programming and parameter setting. A hierarchical, compositional programming language, *Corelet*, is available to aid the development of TN applications [50]. IBM provides support and a development system as well as "Compass" a scalable simulator [51]. The software environment runs under standard

Linux installations (Red Hat, CentOS and Ubuntu) and has standard interfaces to Matlab and to Caffe that is employed to train deep neural network models. The TN architecture can be interfaced using native AER to several bio-inspired sensory devices developed over many years of neuromorphic engineering (silicon retinas and silicon cochleas). In addition the architecture is well suited for implementing deep neural networks with many applications in computer vision, speech recognition and language processing.

Loihi

The Loihi chip [52] was designed by Intel Labs, and consists of a 60mm^2 die, 2.07 billion transistors, and 33 MB of SRAM fabricated in Intel CMOS 14nm. Its digital architecture is comprised of 128 neuromorphic cores, each with 1024 spiking neural units (compartments), and claims to implement novel features such as hierarchical connectivity, synaptic delays, and on-chip learning. Each core is connected via an asynchronous mesh Network-on-Chip (NoC) and includes three embedded 32-bit processors. The NoC allows core-to-core multicast communication, variable synaptic formats, and population-based hierarchical connectivity. Peripheral interfaces allow the system to communicate with other similar chips via a hierarchical packeting system, thus allowing an overall expansion of the spiking neural network, reaching up to 4096 cores/chip, and 16384 chips. The NoC implements a request and acknowledge handshaking protocol allowing the CPU cores to perform read/write operations on the network, as well as set configuration parameters. Each core can implement up to 1024 time-multiplexed neurons, whose structure is embodied by a 2Mb data memory, which holds neuron states and synaptic parameters. A synaptic unit processes afferent spikes and reads their associated weights from memory. A dendritic unit updates neuron state variables, an axon unit generates efferent spikes for all fanout cores, and a learning unit utilizes preconfigured learning rules to update the synaptic weights.

Loihi has 16 MB of synaptic memory, providing 2.1 million synaptic parameters per mm^2.

Other Work

Other experimental work from recent publications explore different approaches to the idea of processing in memory [53–59].

2.5 Discussion

This chapter addresses the shortcomings of conventional computing, specifically in Von Neumann architectures where the arithmetic-logic units are physically separated from main memory, resulting in a huge expenditure of energy for shuttling data. Compute-in-Memory is introduced as a solution to the memory-bottleneck; the solution is to embedded processing into regular memory crossbar arrays that naturally lend themselves for efficient computing in space, energy, and time. Next, analog vs. digital CiM architectures were discussed, showing the advantages of both configurations; in this thesis, analog and mixed-signal computing is chosen based on the promising energy and area density savings that both DRAM and NVM possess. Noteworthy, the crossbar architectures discussed were left general and apply to both NVM and DRAM technologies which are discussed in Chapters 3,4, 5, and 6.

The general memory crossbar architecture is discussed and showed to be useful for computing vector-matrix multiplications, a ubiquitous task in Artificial Intelligence (AI). The naive multiplication algorithm and its CiM hardware mapping is showcased. The advantage of binary coded computing lies in the storage density, where a single bit represents the weighted value of for the corresponding digit. However, the increase in density comes at the expense of increased post-processing hardware, and the vulnerability to interference.

Next, unary computing was introduced with the motivation of increased interference resiliency. In the scope of unary computing, both Pulse-Width-Modulation and Stochastic Computing was shown to map to the CiM hardware at hand. In both cases, a scalar-vector and a vector-matrix multiplication is exposed, showing the trade-offs between space and time.

Finally, a brief description of the state-of-the-art of Process-in-Memory Application Specific Integrated Circuits (ASICs) is shown; (i) the IBM TrueNorth chip, which

consists of 4096 cores, implementing 256 neurons (processor/memory) that operate on a unary data representation, and (ii) the Intel Loihi chip, which is also a multi-core neuromorphic chip that multiplexes 1024 units (processor/memory) which use unary-synaptic data representation as well.

Chapter 3

Non-Volatile Primitives

3.1 Introduction

In the year 2013, the Defense Advanced Research Projects Agency (DARPA) started the Unconventional Processing of Signals for Intelligent Data Exploitation (UPSIDE) [60] program. The goal of this program was to ameliorate existing technologies for real-time processing of high-resolution video imagery, specifically for applications such as object tracking and target recognition. The former, usually involves operations such as pattern matching and convolutions. In order to address these operations, this work explores different Compute-In-Memory architectures and technologies that enable high throughput and low power algebraic operations.

Particularly, this chapter explores the usage of flash Non-Volatile Memory (NVM) [61] due its capability of storing either binary, or analog data [62] in the form of trapped charge on a floating gate of a MOSFET, even when the device has been powered down. Furthermore, floating gates have been scaling-down consistently over the years [63], making it to 28nm embedded technologies, which enables NVM integration with logic CMOS processes and opens the door to in-memory computing. The NVM IP used in this work is the ESF3 Split Gate SuperFlash® [64], provided by Silicon Storage Technologies (SST), a subsidiary of Microchip. This IP was licensed for academic use within the DARPA-UPSIDE program.

In order to address the feasibility questions of NVM technology, the first part of this chapter focuses on the design of a small two-by-two floating-gate array, which was fabricated for characterization and modeling purposes. Next, both analog and digital models were used to prove usefulness in a standard image processing task. The second part of this chapter shows how the cells are integrated into a partially programmable convolution processor that is integrated with custom digital logic. Finally, results and conclusions are shown.

3.2 A Two-by-Two Floating-Gate Array in 55nm

A two-by-two floating-gate array was designed and fabricated with the following goals in mind:

- Demonstrate multi-level storage (as opposed to binary), thus increasing both storage and computational density.
- Utilize test data to create both analog and digital models that are useful for simulating behavior at the system level.

3.2.1 Cell Operation

Table 3.1 shows typical array operation [64], and Fig. 3.1 shows the cross section and schematic of a pair of devices. The word-lines (WL) are used to activate series access-transistors, which create an "ANDing" effect with the stored charge. Typical arrays are built by connecting two devices in parallel at the physical level, forming a NOR flash-cell. Each NOR-cell shares: erase-gates (EG), source-lines (SL), coupling-gates (CG), and bit-lines (BL). Word-lines (WL) are routed individually, allowing cell-level program/read granularity. Figure 3.2 shows schematics for each mode of operation, based on Table 3.1.

Table 3.1 *Standard Memory Cell Operation*

	CG		SL		EG		WL		BL	
Mode	sel	uns	sel	uns	sel	uns	sel	uns	sel	uns
Read	2.5	2.5	0	0	0	0	2.5	0	0.8	0
Erase	0	2.5	0	0	HV	0	0	0	0	0
Program	HV	2.5	4.5	0.5	4.5	0.5	VWLP	0	VPR	2.5

*HV $\geq$ 8[v]

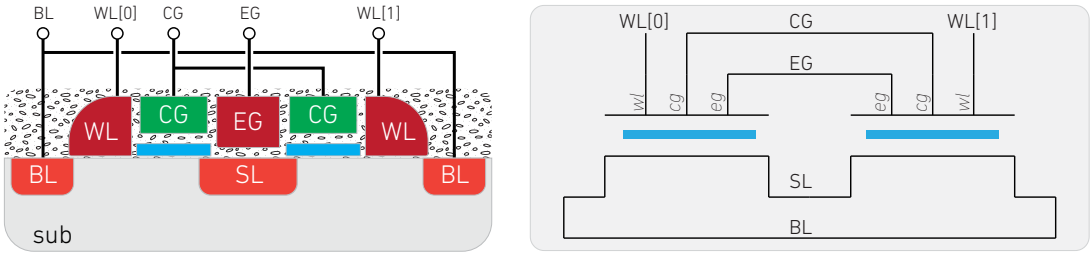
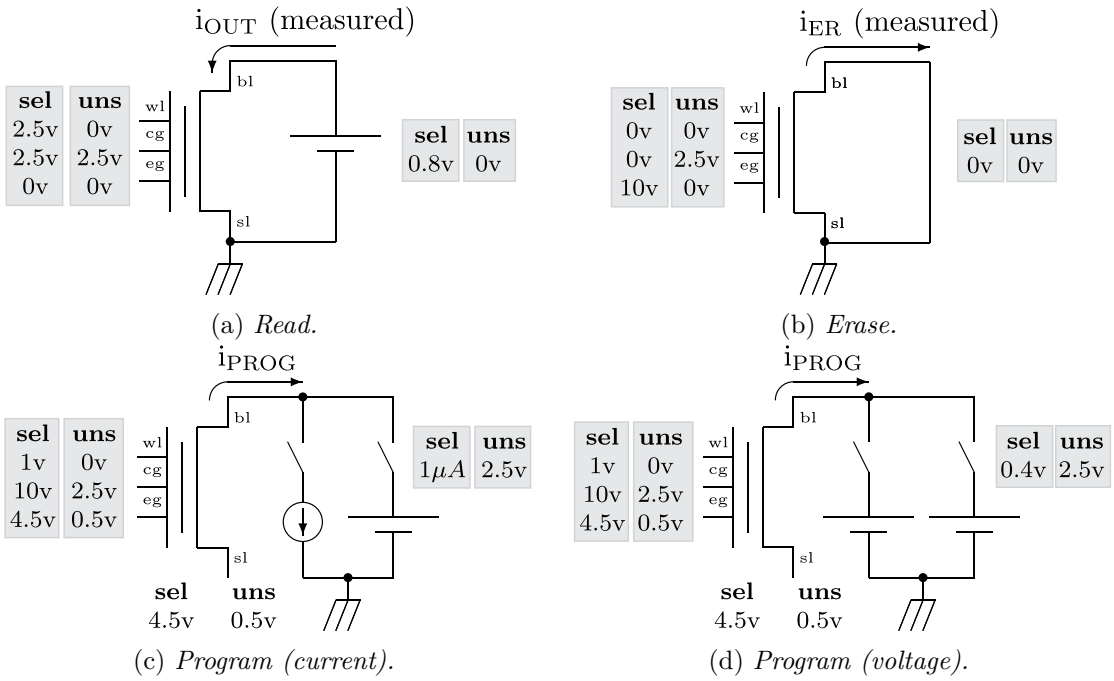


Figure 3.1 *Cross-section and schematic of a NOR flash cell.*



(a) *Read.* (b) *Erase.*

(c) *Program (current).* (d) *Program (voltage).*

Figure 3.2 *Floating gate operations:* ***(a)*** *read,* ***(b)*** *erase, and* ***(c)***,***(d)*** *program. Notice that each sub-figure shows the necessary voltages, or currents, used to perform each operation. Notice that there are two distinct ways to perform programming: the first, by applying current from source-line to bit-line ($I_{SL,BL}$), and the second, by applying a positive voltage across the source-line to bit-line channel ($V_{SL,BL}$).*

Read Operation

The read operation consists of enabling a device by supplying a positive voltage to the word-line. A drain-source voltage ($V_{BL,SL}$) biases the device and allow to read the total current though the bit-line.

Program Operation

During the program phase, a high electric field across the drain-source region causes hot-carrier injection [65] across the gate oxide. This is possible due to the high voltage applied to the coupling-gate at the time. The injected charge remains trapped under the floating gate, causing the effective threshold to drop. A fully programmed cell creates a threshold shift that is large enough to cause a zero current during read-out. Moreover, if programming is carried out gradually, then intermediate states may be achieved, thus enabling multi-level storage functionality.

Erase Operation

During the erase phase, any trapped charge under the floating gate is removed via Fowler-Nordheim tunneling [65]. This is achieved via a high voltage on the erase-gate, which is physically near the floating-gate. Consequently, a fully erased cell's threshold goes down, enough to produce currents up to $\sim 37\mu A$. It is important to note that in this work, the erase operation was only used to clear all the charge under the floating gate, i.e., there is no gradual erase procedure that would allow multi-level storage in this particular technology.

3.2.2 Array Operation

NOR flash-cells are arrayed to form a cross-bar, such as shown in Fig. 3.3. Array operation follows Table 3.1, where pairs of rows (also referred to as *pages*) may be erased, or individual cells may be programmed and read. When several cells are enabled simultaneously and they share a common bit-line, the read-out current is added linearly. If the charge stored under the floating-gates represents a linearized weight, then the total current on the j^{th} bit-line is:

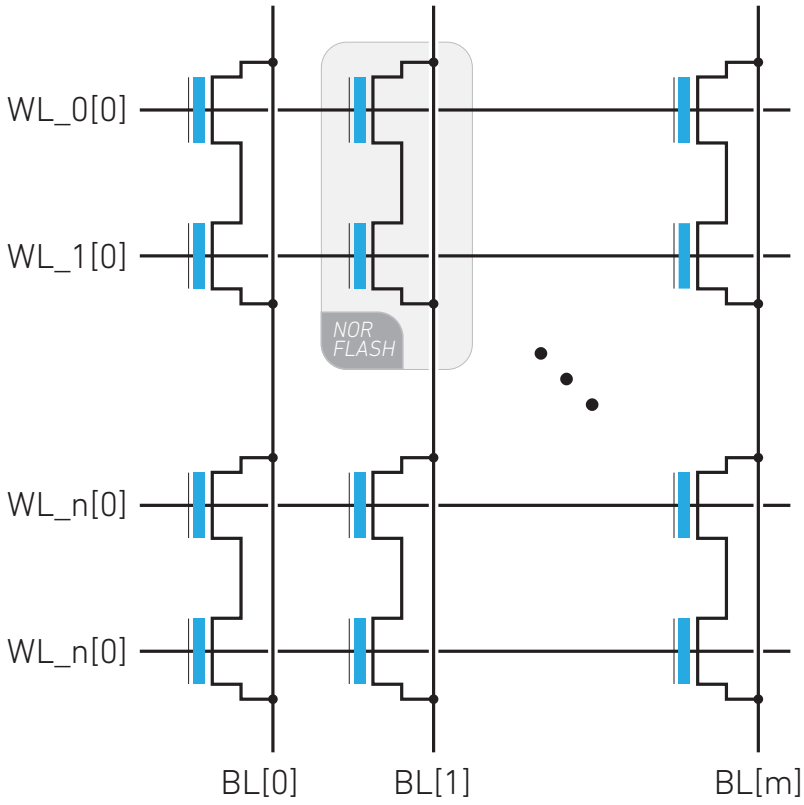


Figure 3.3 *NOR flash-cells, arrayed forming a cross-bar.*

$$I_{BL_j} = \sum_i x_i \, w_i, \tag{3.1}$$

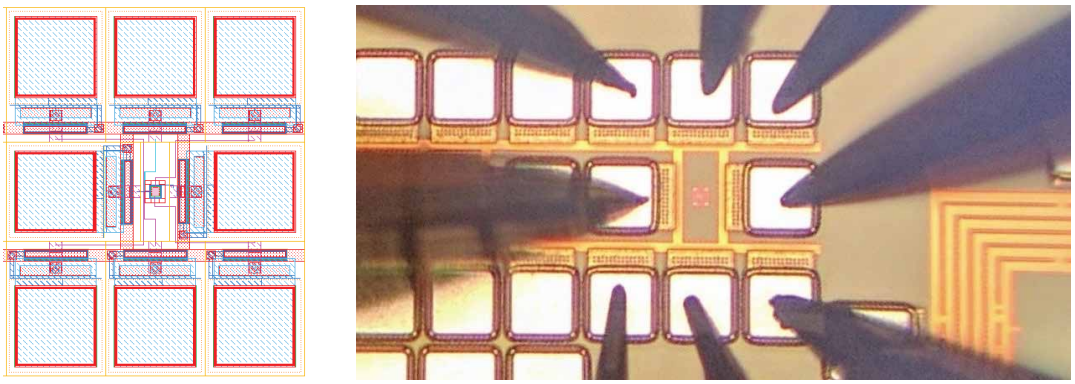
where x_i represents an the i^{th} word-line, and w_i is a linear mapping between between the stored charge and the read-out current a half-flash-cell.

3.2.3 Experimental Results

A 2x2 flash-cell array was fabricated in CMOS 55nm. Due to matching purposes, and in order to avoid edge-effects at the peripheral of the array, several dummy rows and columns were added. The layout and micrograph are shown in Fig. 3.4.

Test Setup

The test setup is shown in Fig. 3.6, and consisted of utilizing three source-measure-units, one quad voltage supply, and a microscope probe station. All instruments were



(a) *Layout.* (b) *Micrograph.*

Figure 3.4 ***(a)*** *Layout and* ***(b)*** *micrograph, of 2x2 floating-gate array. The figure on the right shows probes that were used to interface the chip with source-measure-units (SMUs).*

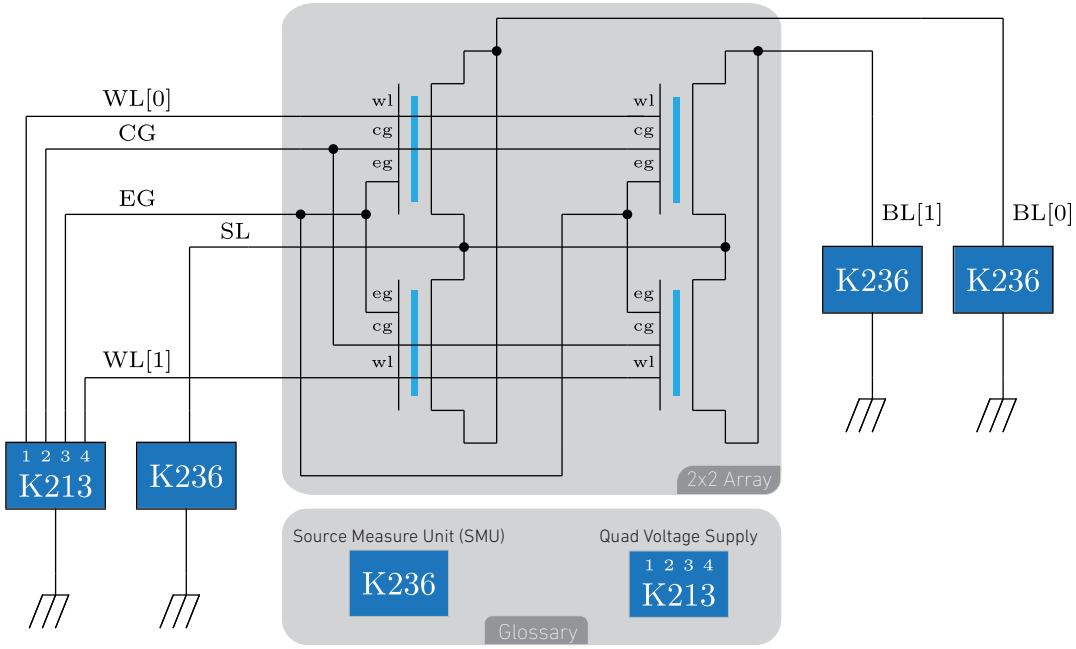


Figure 3.5 *Schematic of a 2x2 floating-gate test setup. The actual array size is 20x20, with 8 rows/columns of grounded dummy-cells on the periphery of the array.*

controlled via GPIO and programmed via MATLAB.

Input Transfer Curves: *gm*

Figure 3.7 shows test results for different cells programmed at different levels. After an erase operation, each cell was targeted individually, and later gradually programmed, followed by a read operation. This iterative process was done until the desired current

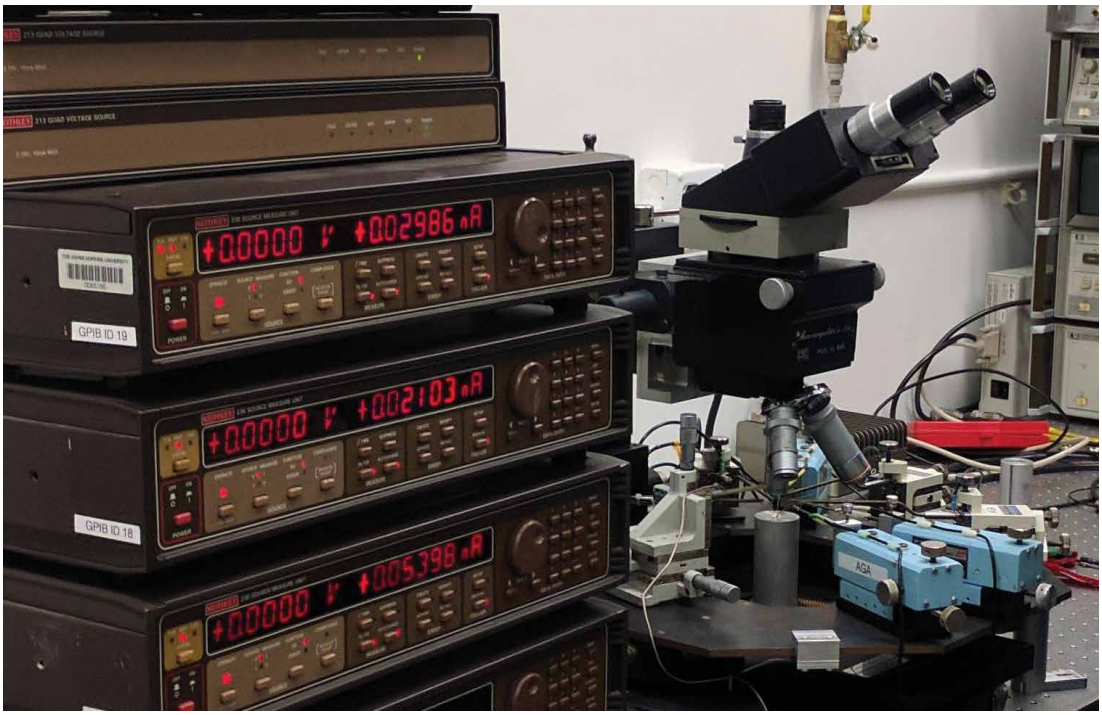


Figure 3.6 *Floating gate test setup with probe station. The instruments on the left are source-measure-units (SMUs) and voltage supplies, used to generate all necessary control signals.*

amplitude was achieved. Next, a word-line voltage (V_{WL}) sweep at constant drain voltage (V_{BL}) was carried out, representing the effective transconductance of the device. Typical cell operation takes place at $V_{WL} = 1$v, yielding a maximum output current per cell of approximately $10\mu A$. Noteworthy, there is a change in the current direction between the read and program phases, which forces an iterative program-read cycle.

Output Transfer Curves: *gd*

Figure 3.8 shows the output transfer curves for individual cells that were programmed as described in the previous section. Output curves were plotted in three steps:

1. Drain voltage (V_{BL} is swept for different gate voltages on the i^{th} row. In the case of a 2x2 array, $i = 0$. All other word-line voltages were set to zero, thus disabling current outputs from non-selected cells). This is illustrated in the first row of Fig. 3.8.

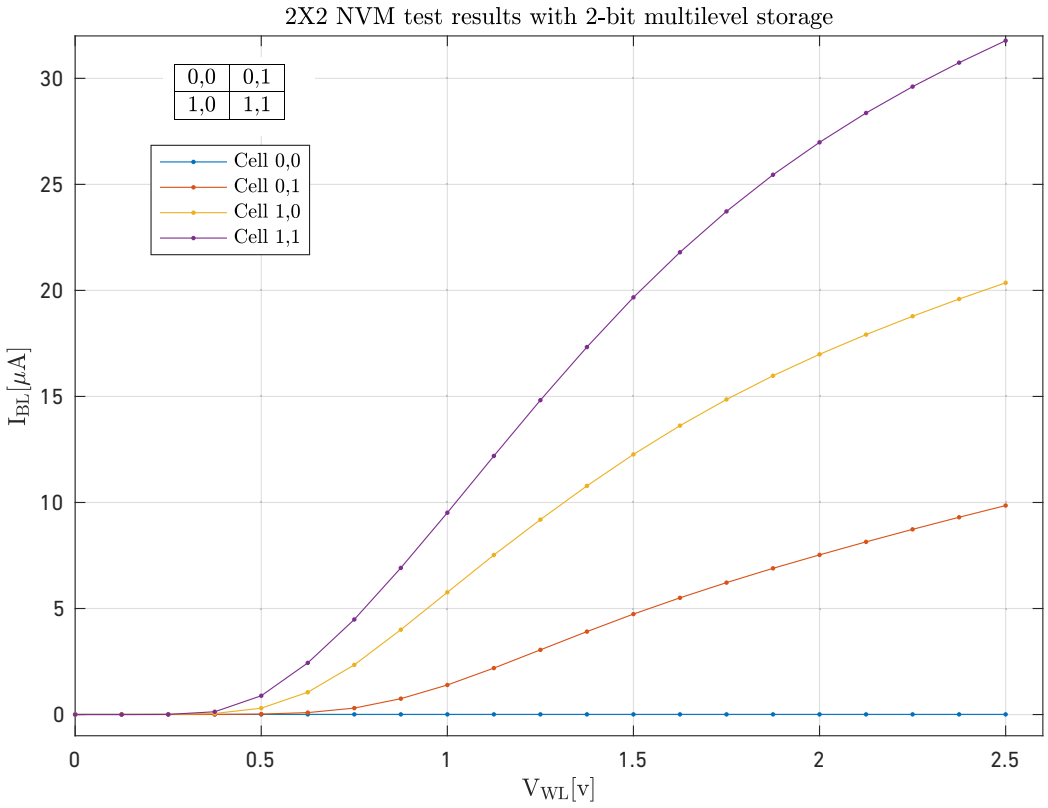


Figure 3.7 *Input transfer curves for the the 2x2 array: the four bit-cells were programmed with four different currents which results in four distinct transfer curves. The WL voltage was swept from 0 to 2.5V while the BL current was measured for each bit-cell. During read operations, the CG, EG, and SL of unselected and selected cells were held at the same potential. BL current with WL voltage @ 2.5V is the nominal read current for each programmed cell.*

2. The previous step is repeated for all other rows. This is illustrated in the second row of Fig. 3.8.

3. The previous step is repeated but enabling all word-lines. Test results show that currents are linearly added on the bit-line node, therefore demonstrating the ability of the array to perform vector matrix multiplication (VMM). This is illustrated in the third row of Fig. 3.8.

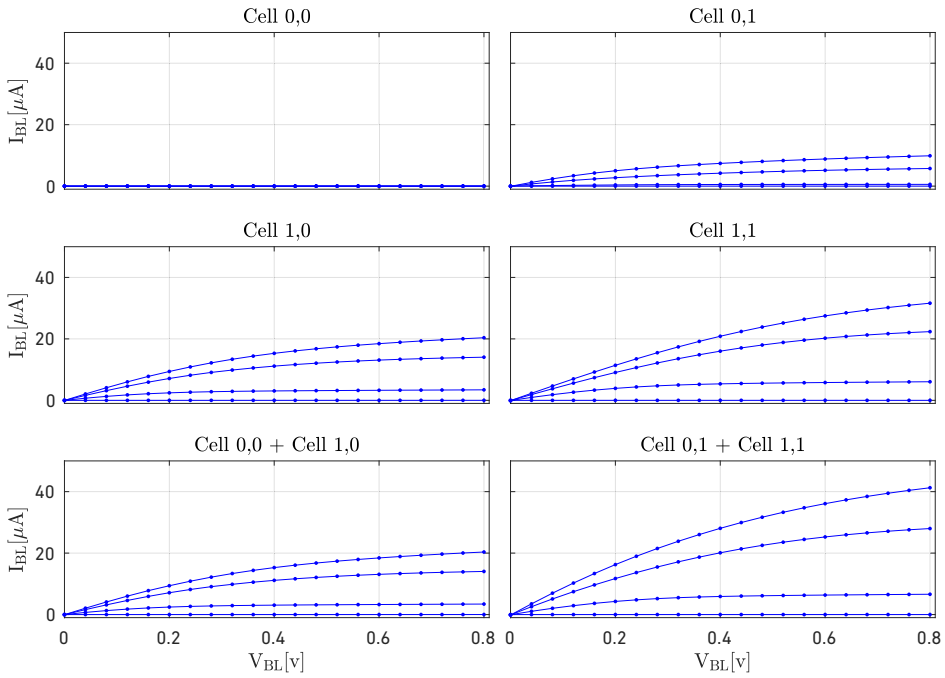


Figure 3.8 *Output transfer curves for the 2x2 array: the BL voltage is swept using the K236 SMU while simultaneously reading the BL current for each bit-cell and column. The top four plots depict the stated output transfer relation for each bit-cell while the bottom two plots show the cumulative current in each column. The family of curves in each plot was generated by sweeping the WL over four distinct levels.*

3.3 Models and Simulations Based on Test Data

The goal of the previous section was to establish baseline test data such that floating-gate models could be developed for usage in larger system emulation and design. In this section, two models are presented:

- A Verilog-AMS model based on the unified metal-oxide-semiconductor field-effect-transistor (MOSFET) equations [66]. These continuous equations encapsulate both long and short-channel effects, and are shown below:

$$I_D = \begin{cases} k' \frac{W}{L} \left(V_{GT} - \frac{V_{min}^2}{2} \right), & V_{GT} \geq 0 \\ 0, & V_{GT} < 0, \end{cases} \tag{3.2}$$

  where $V_{GT} = V_{GS} - V_T$, $V_{min} = \min(V_{GT}, V_{DS}, V_{DSAT})$, and V_{DSAT} is the drain-source velocity saturation voltage.

- A digital synthesizable accumulator, which models the total current that a cell can output. The value of the accumulator can be changed via digital logic.

3.3.1 Verilog-AMS Model

A Verilog-AMS model based on Eq. 3.2 was created, where the free parameters where used to fit test data to the model. The Verilog-AMS model uses special *electrical* net types, where nodes are defined to obey Kirchoff's circuit laws. This model enables the construction of large arrays, where currents are added linearly on nodes, making it physically plausible. Figure 3.9 shows the input transfer curves of the Verilog-AMS model after fitting parameters to test data.

Test Bench

In order to functionally verify the Verilog-AMS (VAMS) model, a mixed signal test bench was established in the Cadence mixed-signal environment. The test bench

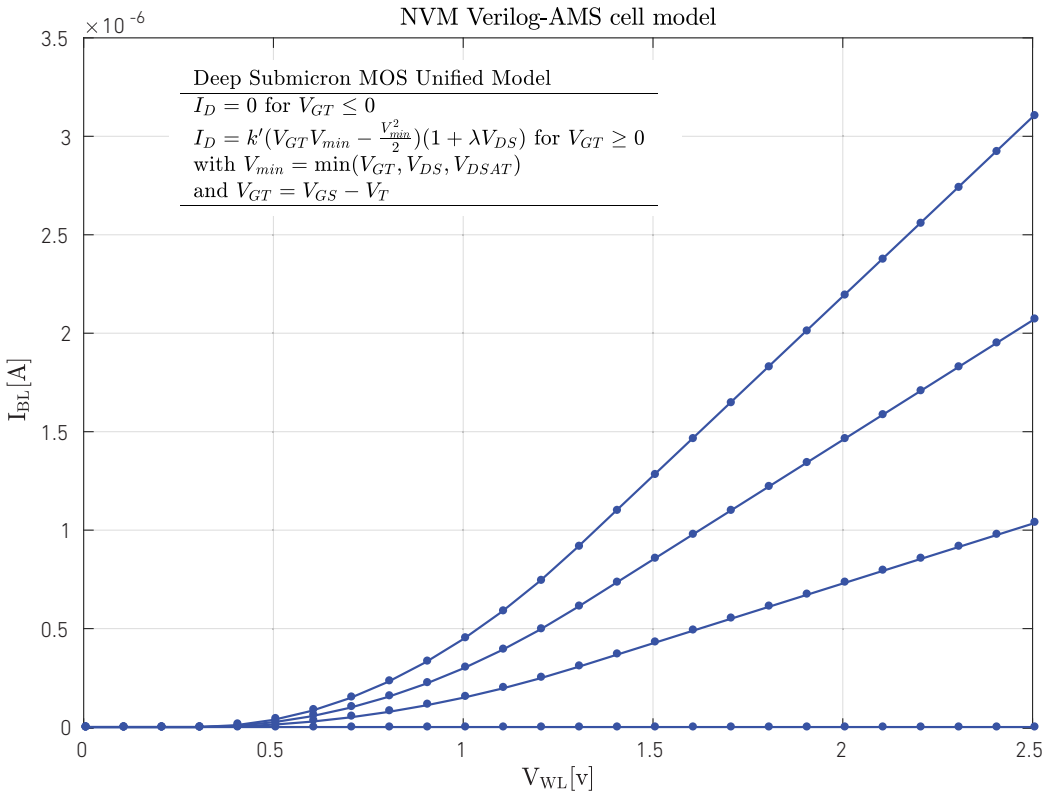


Figure 3.9 *Floating gate model based on MOSFET unified equations.*

(a) (b) (c)

Figure 3.10 ***(a)*** *Original cameraman image with 4-bit resolution per pixel.* ***(b)*** *Cameraman filtered with Sobel kernel in MATLAB. Sobel kernel was quantized to have 2-bit parameters, yielding a 7-bit output image.* ***(c)*** *Cameraman filtered with Sobel kernel with Verilog-AMS in the Cadence environment. Sobel kernels were stored in two columns in order to store negative weights.*

instantiates the VAMS model of the floating gates in the form of an array, where bit-lines are connected vertically, and word-lines are connected horizontally. Input data and floating-gate program values are loaded via text files that are pre-processed in MATLAB. Similarly, output processed data is saved in text files for viewing later in Python or MATLAB.

Simulation Results

The test bench loads a Sobel kernel and a cameraman image from text files into the test bench. The Sobel kernel is mapped to a real number and used as an *analog memory* to weigh the outputs of individual floating-gates during read-out. Negative weights are stored on separate columns, and results are subtracted later to achieve the desired processing. A state machine reads the text file containing the input image and maps digital values to real-valued signals and wires them to word-lines in the array. As image patches are processed, they are saved to an output file and later displayed in MATLAB. Figure 3.10 shows the original image, and the result of applying a Sobel kernel filter, both in MATLAB and Verilog-AMS simulation.

3.3.2 Digital Model

The VAMS model brought insight into how a floating-gate array could be constructed and utilized to compute convolutions in the analog domain. From a practical standpoint, computing in the analog domain requires two additional blocks:

- A pre-compute digital-to-analog conversion (DAC or D/A).
- A post-compute analog-to-digital conversion (ADC or A/D).

From a system level perspective, the computational unit is a digital-input, digital-output block. Therefore, the fact that the computation is carried out in the analog domain is not relevant from the perspective of high-level system simulations. Hence, a digital read/write/compute model was developed, simulated, and deployed on an FPGA. The model captures a high level of abstraction, where analog values no longer exist. Nonetheless, the model does capture some physical properties of the floating-gates, such as multilevel storage, summation of currents on a node, and accounts for both D/A, and A/D conversions.

Model Description

Figure 3.11 shows the circuit implementation of the NVM periphery controlling the j^{th} column and its high-level abstraction model used in simulation. From the left-hand side of the figure, *ROW* signals represent inputs, X, which control level-shifters, which in turn control the NVM column. Finally, the bit-line current is wired to the sigma-delta for A/D conversion. From the right side of the figure, the bit-cell state is modeled by an accumulator of M-bits. When multiple rows are enabled during a read operation, the output of all accumulators are added and wired to input of the sigma-delta model (shown in Chapter 5). During program operation, only one row at a time is enabled, thus coupling-gate (CG) pulses cause the corresponding accumulator to decrement by 1. Similarly, a burst of R pulses will cause the accumulator to decrement by R. This model applies directly to the programming algorithm described in previous sections, where the target cell is iteratively read and programmed until the desired value is reached. Moreover, the erase operation is simply carried out by resetting accumulators to their maximum value, which represents the maximum current value per NVM bit-cell. A detailed description of the model's behavior is described below:

- **Digital to analog conversion:** The DAC is modeled by a pulse-width-modulator, which consists of comparing binary valued inputs with a monotonic ramp. For each input timeslot, a unary vector is wired to the word-lines of the array.

- **Analog to digital conversion:** The ADC block consists of a first order Sigma-Delta modulator, which re-quantizes and integrates data along columns. The re-quantization process in the digital domain emulates the actual A/D conversion that takes place in the true mixed-signal space. The choice of A/D converter was based on power and area estimates and will be described in future sections.

- **Erase:** The elemental storage block in the model consists of an accumulator,

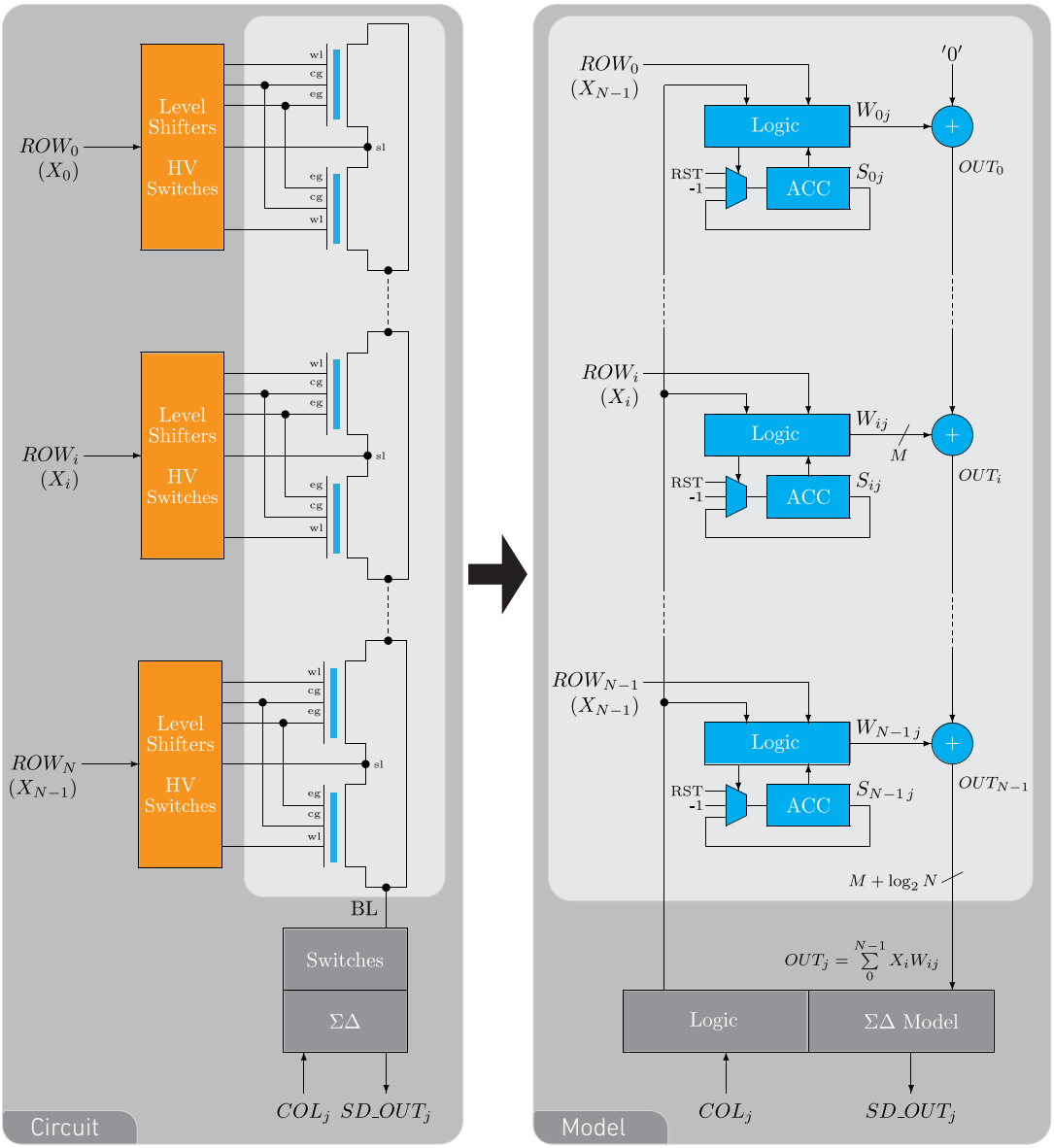


Figure 3.11 *Non Volatile-Memory digital write/compute model. The figure shows the model for the j^{th} column.*

which is reset to its maximum value during the erase phase. Under these conditions, the Sigma-Delta modulator re-quantizes the digital value to match its maximum range for an erased column.

- **Program:** During programming, the accumulator is decremented by pulsing the coupling-gates. These pulses control the select-input to the multiplexer in each cell. For each pulse (or group of pulses), the output data is read-out and compared to a reference value. This iterative process is repeated until the read-out value reaches the reference and has been designed to embody the physical operation of the analog array, which was described in previous sections.

- **Compute:** During the compute phase, a time-domain PWM signal is applied to the word-lines. For each timeslot, the stored values in each accumulator are added and re-quantized by the Sigma-Delta. Next, subsequent PWM values are updated and the Sigma-Delta continues to re-quantize and accumulate the result. Once the operation has terminated, the user may read-out the result of the computation.

3.3.3 FPGA Emulation Model

In order to verify the validity of the digital model, a high-level abstraction was implemented in an FPGA and used to demonstrate Vector-Matrix Multiplications (VMMs). In order to accelerate throughput, the model used in the FPGA omits the Sigma-Delta modulator and takes a scaled output as a result. The main computation block is illustrated in Fig. 3.12.

Vector Matrix Multiplier Architecture

As the name suggests, a VMM architecture computes the multiplication of an input vector, $\mathbf{X} \in \mathbb{R}^n$, with a stored matrix of weights, $\mathbf{W} \in \mathbb{R}^{n \times m}$ resulting in $\mathbf{Y} \in \mathbb{R}^m$. In short, the operation takes place as a result of successive read operations from

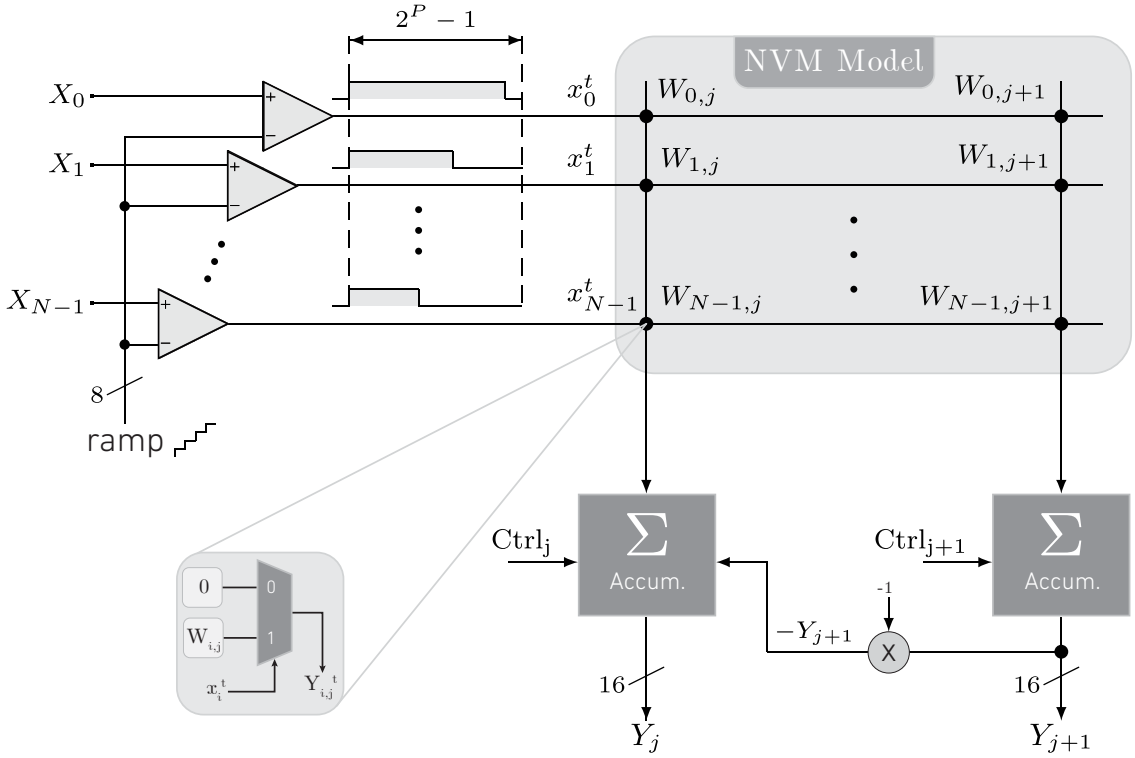


Figure 3.12 *Non-Volatile Memory digital model deployed in an FPGA.*

programmed NVM bit-cells which each hold an element of matrix $\mathbf{W}$, $W_{i,j}$, that has a pre-determined resolution, M bits. Each of the n-elements of $\mathbf{X}$ is encoded via pulse-width modulation (PWM) that results in a unary stream x_i^t of length $2^P - 1$, where P is the desired resolution of X_i and t denotes the t^{th} element in x_i. Formally, to compute, the output current of each cell using this unary stream, x_i, is:

$$I_{i,j}^t = Y_{i,j}^t = \begin{cases} W_{ij}, & x_i^t = 1 \\ 0, & x_i^t = 0 \end{cases}$$

As seen in Fig. 3.11, the digital representation of the currents on the bit-line are accumulated, so the result over the n-rows of the NVM array can be seen as:

$$Y_j^t = \sum_{i=0}^{N-1} Y_{i,j}^t = \sum_{i=0}^{N-1} W_{i,j} x_i^t \tag{3.3}$$

To complete the full computation, this output is computed for $2^P - 1$ iterations, where unary value x_i^t changes depending on the input element X_i. Therefore, the expression

becomes:

$$\mathbf{Y}_j = \sum_{t=0}^{2^P-1} \mathbf{Y}^t = \sum_{t=0}^{2^P-1} \sum_{i=0}^{N-1} W_{i,j} x_i^t \tag{3.4}$$

Now, the resolution of the accumulated column-wise output, $\mathbf{Y}_j$ is $\log(N) + M$ bits, and represents one element of the unsigned one-quadrant VMM. In order to expand to two quadrants, $W_{i,j} > 0$ can be programmed on column j and $W_{ij} \leq 0$ are stored on column $j+1$, by programming its absolute value, $|W_{i,j}|$. By subtracting the two results, $\mathbf{Y_j}$ and $\mathbf{Y_{j+1}}$, the two-quadrant VMM, $\mathbf{Q} \in \mathbb{R}^g$ where $g = \frac{N}{2}$, is realized:

$$\mathbf{Y}_j = \sum_{t=0}^{2^P-1} \sum_{W_{i,j}>0} W_{i,j} x_i^t \tag{3.5}$$

$$\mathbf{Y}_{j+1} = \sum_{t=0}^{2^P-1} \sum_{W_{i,j}\leq 0} |W_{i,j}| x_i^t \tag{3.6}$$

$$\mathbf{Q_r} = \mathbf{Y}_j - \mathbf{Y}_{j+1} = \sum_{t=0}^{2^P-1} \sum_{i=0}^{N-1} W_{i,j} x_i^t \tag{3.7}$$

Practical Considerations

- **Emulation:** The embedded flash memory is emulated via register-file, whose contents are loaded prior to implementation. This initialization took the place of the multi-level programming of the NVM array. For implementation on the FPGA, the read operation of the bit-cell is behaviorally modeled with a register and multiplexer. When the unary bit x_i^t is asserted, $W_{i,j}$ is selected, which is in accordance with Section 3.3.3.

- **Digital periphery:** To implement an FPGA-based emulation of the NVM-VMM, based on the conversation in Section 3.3.3, the PWM encoders and accumulators are needed. Depicted in Fig. 3.12, the encoding was accomplished via comparison, using a simple ramp counter of identical resolution to the input, resulting in $x_i^t = 1$ when $X_i \geq C$, and 0 otherwise, where C is the ramp counter value. The encoding period is configurable and determines the effective input

data resolution. Output accumulators, labeled by the output vector element $\mathbf{Y_j}$ in Fig. 3.12, are enabled once every $2^P - 1$ compute cycles, and added to $\mathbf{Y_j}^{t-1}$, from previous cycle's result. In addition, a simple state machine controlled the N input encoders and enabled the output accumulators.

- **System Integration** Ultimately, the NVM-VMM can be used as a hardware accelerator adjacent to the central processing unit or node-purposed for distributed computing. To demonstrate this capability, a miniature NVM-VMM system is devised and implemented on an FPGA, consisting of the following blocks: A simple instruction set processor, a control unit, and the NVM-VMM. To facilitate this flow and allow for integration in existing multiprocessor systems, the NVM-VMM and ICU are equipped with features from the ARM protocol, AMBA 3 AHB-Lite bus, a two-phase transaction protocol in a master-slave configuration [cite arm manual]. A simple assembly-like program can be written to describe system operation. Once compiled, the data and periphery configuration is dictated by the instructions stored in memory. Pre-processed data is stored in a FIFO, which is read by the control unit and shepherded to the NVM-VMM. Once a VMM has been completed by the slave unit, a request is sent to the control unit, and the data is sent to and output FIFO for retrieval.

Results

The FPGA emulation of the NVM-VMM is implemented on a Xilinx Kintex-7 (XC7410T) and communicates through an Opal Kelly 7350 FPGA Board via USB 3.0 interface. The SoC is configured using parameters detailed in Table 3.2a, to process 128x128-pixel images sent via USB to the FPGA. To note, the organization of data dictates the size of the array and throughput of the system, leading to the dimensions listed in Table 3.2a. Two-quadrant VMM is demonstrated by storing positive and negative kernel weights on separate columns, and later subtracting the outputs. This

Table 3.2 *NVM-VVM FPGA implementation:* ***(a)*** *parameters and* ***(b)*** *performance [62].*

(a)

Parameter	**Val.**
Bit-cell precision (l bits)	3
Patch Size (p_s)	3×3
# of Patches (n_p)	8
Input resolution (k bits)	4
# of Kernels (n_k)	2
# of Rows ($n = p_s * n_p$)	72
# of Columns ($m = n_p * n_k$)	16

(b)

Metric	**Val.**
Clock Speed	200 MHz
Array dimensions	72 x 16
Memory Capacity	3.4kb
Operation	4-bit MAC
Throughput	1.8 GOPs
Processing Speed	250 fps

is shown in Figs. 3.13d & 3.13c, using the Sobel and Negative kernels, respectively. Similarly, positive valued kernels were used to process the input image using the identity, and Gaussian low-pass kernels, shown in Figs. 3.13a & 3.13b, respectively. Since the system operates on an FPGA, the computational efficiency could not be resolved, but the NVM-VMM throughput was calculated and is presented in Table 3.2b.

(a) (b)

(c) (d)

Figure 3.13 *Examples of two-quadrant VMM kernel processing using FPGA NVM SoC.* ***(a)*** *identity,* ***(b)*** *Gaussian low pass,* ***(c)*** *right Sobel, and* ***(d)*** *Negative.*

3.4 Compute-in-Memory (CiM) Test-Chip Based on Non-Volatile primitives

Digital and analog models described in Section 3.3 were used to develop and create a partially programmable Compute-in-Memory (CIM) test-chip, which was designed and taped-out in CMOS 55nm. The used NVM IP consisted of ESF3 SuperFlash provided by Silicon Storage Technologies (SST), a subsidiary of Microchip. This test-chip was part of an incremental development flow supported by the DARPA-UPSIDE [60] program.

3.4.1 Chip Overview

The goal of the test-chip described in this section was to move up in hierarchy from the bare floating-gate arrays that were described in Section 3.2. Figure 3.14 shows the main architecture of the chip. The design is composed of four basic elements:

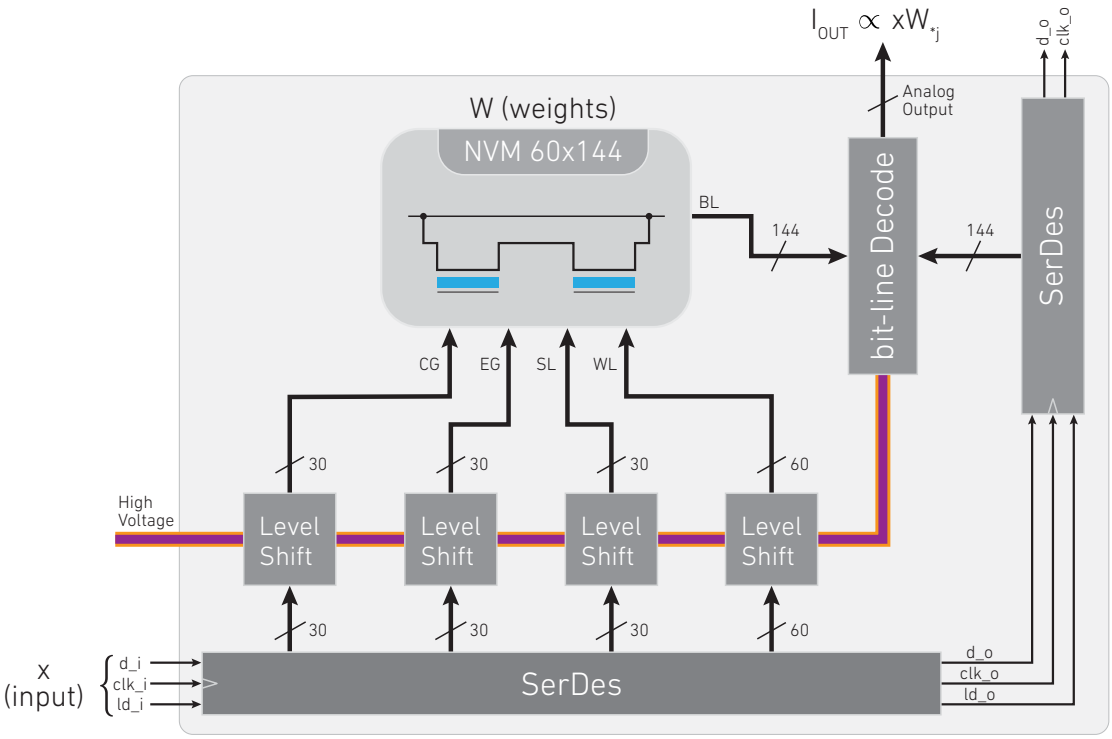


Figure 3.14 *Partially Programmable NVM Compute-in-Memory architecture.*

1. A Serializer/Deserializer (SerDes) bus, which contains parameters for every single on-chip control-line, and is used to shift-in input vectors, x. SerDes data is read in a parallel fashion by tapping into the different bits of the shift register. From the figure, the *load* signal is used to parallel load the input data onto latches, allowing data to be presented simultaneously on the control lines of the NVM peripheral circuits.

2. Level shifters, which take signals from the SerDes and shift them up to the corresponding high-voltage domains. The outputs of switches and level shifters are wired directly to the control signals of the NVM array, i.e., CG, EG, SL, and WL.

3. A 60×144 NVM array, which was constructed in the same way as described in Fig. 3.3. The purpose of the array is to store weight parameters, W, that are later used to perform vector-matrix-multiplications (VMM) against inputs (x).

4. A parallel bank of control switches that are connected to bit-lines (and controlled via SerDes) and allow output array current to be read. This current is proportional to the VVM that takes place between input vector, x, and weight vector, $W_{*j}^T = \{w_{0,j}, \ldots, w_{N-1,j}\}^T$, therefore,

$$Io_j \propto \sum_i x_i w_{ij}. \tag{3.8}$$

   In order to read all j^{th} outputs, the SerDes control signals must be altered accordingly.

3.4.2 Peripheral Circuits

This section describes the necessary peripheral circuits used to operate a 60×144 ($rows \times columns$) NVM array. The main challenge that NVM control circuits pose, is that in order to create tunneling and hot-carrier injection, high on-chip voltages

are necessary. The implication is that the the process will require thick oxide and lightly doped source/drain diffusion FETs. Such devices will be referred to as HV (High-Voltage) FETs. Moreover, the technology imposes other design constrains on HV devices, listed below:

- No triple-well/twin-tub options.
- High-voltage FETs do not support $VGS > 3\text{v}$ [1].
- V_{DS} slew rates for the erase ramp must be limited to $S_R < 110\text{mV}/\mu\text{s}$.
- High voltage devices can only switch at low voltage.
- The transient current on devices with a high drain-source voltage ($V_{DS} > 8\text{v}$) must be limited to $I_{DS} < 200\mu\text{A}/\mu\text{m}$ and $V_{SB} < 0.5\text{v}$.
- High voltage devices must not operate in static mode with $|V_{GS}| > V_T$.
- High voltage devices in HV saturation must have a static $I_{DS} < 20\mu\text{A}/\mu\text{m}$.

High Voltage Level Shifter

In order to comply with the aforementioned design constrains, a special level shifter was designed, and is shown in Fig. 3.15. The circuit is composed by three stages: control, memory, and driver; the control stage, reads and decodes data from the SerDes register; the memory stage consists of a variable power-supply SR-latch; and the driver stage buffers the data into the floating gate terminal. In this illustration, the erase-gate (EG) needs to be driven from zero volts, to HV, while maintaining a $V_{SG} < 3\text{v}$ for the driver's HV-PFET. This is accomplished in the following steps:

- The data from the SerDes register is stored an in SR-latch that lays in a variable high-voltage domain. This is achieved via a special level shifter that transitions

[1] Later, the foundry confirmed that this is allowed under certain circumstances.

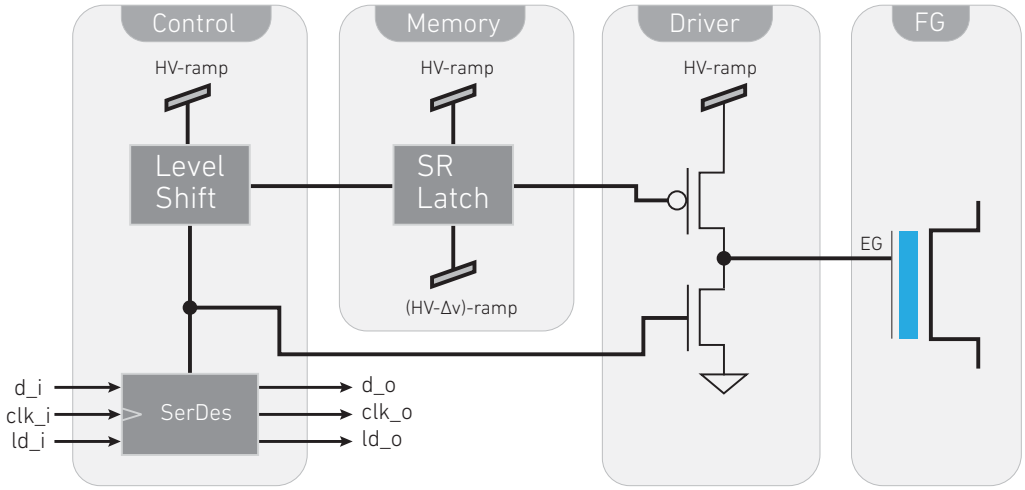


Figure 3.15 *Level shifter scheme for controlling high-voltage nodes in NVM array.*

into a hold-state, i.e. $input = (1, 1)$, while the power supply level is in compliance with the input data.

- Next, the power supply of the SR-latch is gradually ramped up to HV, while maintaining its source-terminals at $HV - \Delta V$. This allows the output of the push-pull driver to be driven to HV while maintaining its source-gate voltage under ΔV.

- After the desired node (EG in this example) has been driven to HV, the inverse process takes place, where HV ramps back down to a low-voltage power supply range.

Read & Program Switches

Another challenge in designing the peripheral circuits for the NVM array is dealing with current-direction changes between the read and the program phases. During the program-phase, current flows from SL to BL. Conversely during the read-phase, current flows from BL to SL, as can be seen in Fig. 3.2. For the purpose of this test-chip, the design approach emphasizes granular control and monitoring of individual nodes via a series of switches, which are shown in Fig. 3.16. The control signals for each individual

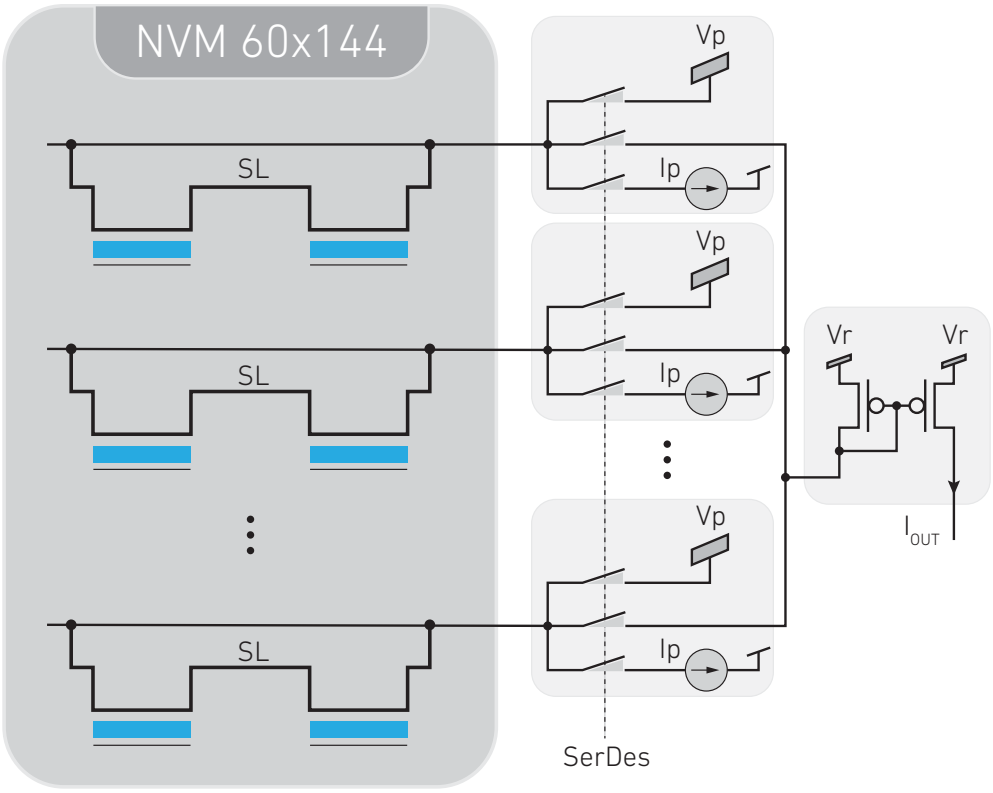


Figure 3.16 *Read and Program switches connected to the bit-line of an NVM array.*

switch are derived directly from the SerDes registers, allowing absolute control and monitoring of the internal nodes. During the read-phase, an arbitrary number of bit-line currents may be added and read-out through a current mirror. Nonetheless, typical operation only requires one switch to be closed at a time, therefore performing the column-wise operation described in Eq. 3.8. Additionally, an extra current mirror was added (not shown in the figure) to optionally amplify the bit-line current by a factor of 76, and read it off chip. This allows for tiny currents ($\sim$ pA) to be captured using, for example, and external A/D converter.

Simplifications

In order to simplify the design, no on-chip charge pumps were incorporated to generate high voltages. Conversely, all high voltage signals and supplies are wired to pads and are expected to be controlled externally. Additionally, no on-chip state machines are

included. The only on-chip memory corresponds to level-shifter latches and SerDes registers.

3.4.3 Results

The top level layout and chip micrograph are shown in Figs. 3.17a & 3.17b, respectively, and consist of two NVM cores, which are connected digitally via a daisy-chained SerDes interface. Each core comprises 60 rows and 144 columns, and the difference resides in the local connectivity of the dummy cells that are located on the periphery of the NVM arrays. The figure on the righ-hand side shows two red rectangles, which correspond to the NVM arrays which are visible due to a metal blockage placed on

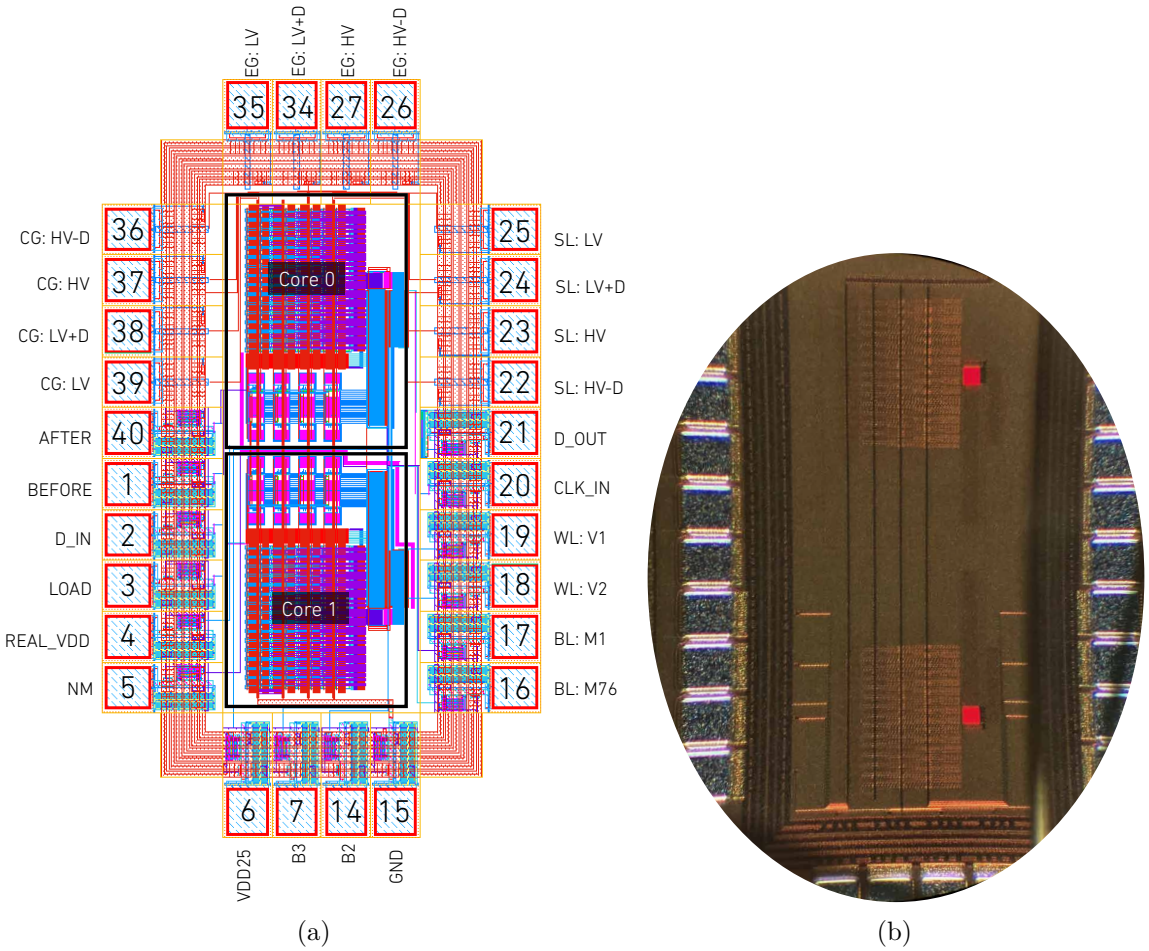


Figure 3.17 ***(a)*** *NVM top layout and* ***(b)*** *micrograph showing two smaller cores connected via SerDes daisy-chain*

the top-level layout; this was done to experiment with UV-light erasing. The top level interface of the chip consists of 40 pins, which are described in Table C.1. The main digital interface consists of a 3-port SerDes (CLK, D_IN, LOAD). In order to minimize power domains across the chip, all digital logic was implemented using 2.5v transistors (medium oxide thickness), as opposed to the low-voltage, 1.2v devices. High voltage transistors (thick oxide) were used in any voltage domain requiring more than 2.5v. Additionally, special pads were designed to withstand high voltage ($\sim 12v$), and ESD discharge.

Core Layout

The core layout and its internal blocks are shown in Fig. 3.18. The challenge at the layout level lies in the size of the high-voltage transistors, where just a few devices are comparable to the full NVM array. Therefore, all control signals at the periphery of the NVM array need to match the pitch of the level shifters, causing a large overhead in wiring. The SR-latches and shifters for the CG and SL are located to the left of the array, and take most of the area, since they are composed of a mixture between medium-voltage (MV) and high-voltage devices. The block on the bottom left is utilized to generate high voltage logic using resistor-based level shifters. The reason for the resistors is to avoid high gate-source voltage drops on gates. The signals generated at this stage are subsequently used to control clamp FETs that are used to generate a hold-state in the SR-latches. These circuits are not critical to the overall operation of the design, and therefore are left out.

Experimental Setup

Figure 3.19 shows the experimental setup for testing the NVM test-chip described in this section. The left hand side of the figure (3.19a) shows the packaged chip (DIP-40) mounted on a custom printed circuit board, which in turn is attached to the FPGA

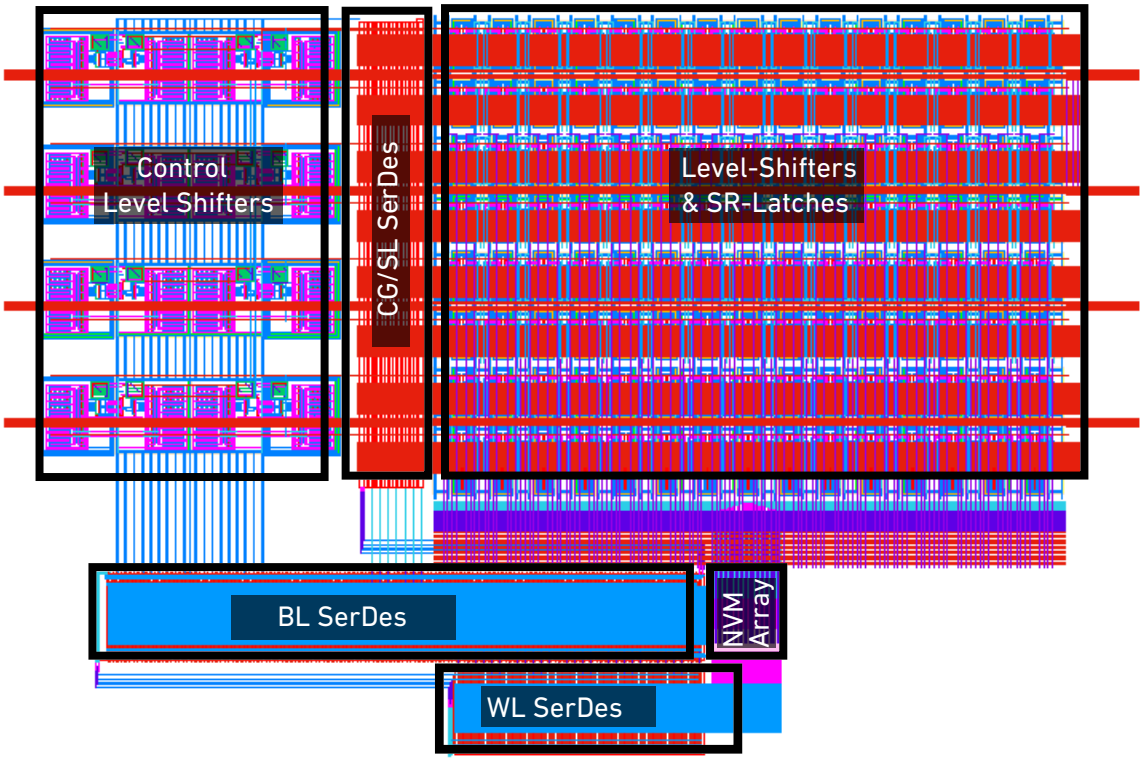
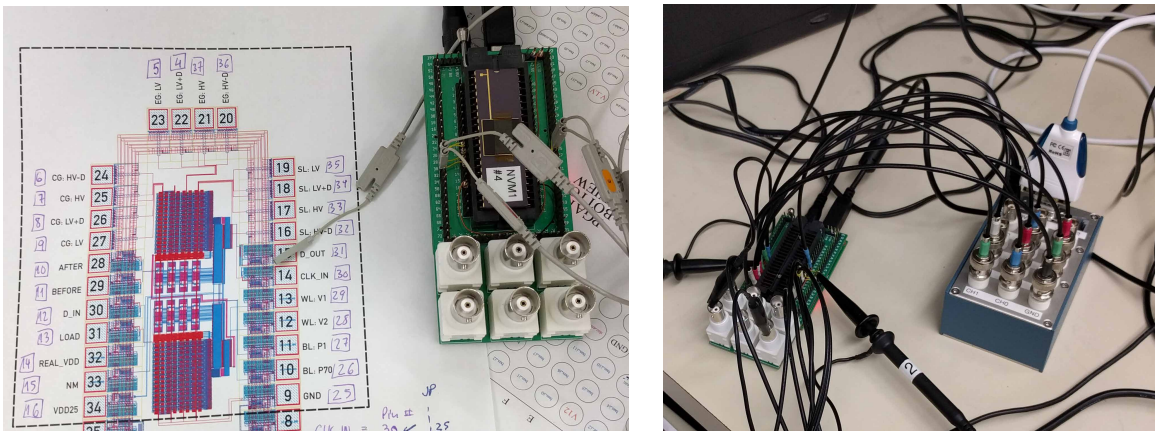


Figure 3.18 *Core layout of 60×144 NVM test-chip. This core is replicated twice and on the top level and programmed via a daisy chained SerDes interface.*

Mezzanine Card (FMC) connector of an Opalkelly XEM6310 board. The former uses a Xilinx Spartan 6 FPGA and connects to a host PC via USB2. The test bench for the system was written in MATLAB via Opalkelly SDK. The figure shows the minimum test setup for testing the digital portion of the chip, which encapsulates the SerDes registers and latches. All digital signals to/from the chip are generated and collected via the FPGA digital input/output ports. Additionally, a logic analyzer was used to debug the interface. The right hand side of the figure (3.19b) illustrates the full test setup, which includes the use of SMUs programmed via GPIO port and controlled through MATLAB, a custom external DAC box (blue casing) controlled via COM port and controlled through MATLAB as well.



(a) (b)

Figure 3.19 *Experimental setup for NVM test-chip array.* ***(a)*** *SerDes digital signals only.* ***(b)*** *SerDes digital signals and analog signals power supplies controlled with external DAC.*

Test Results

Using the test setup described in Fig. 3.6, it was demonstrated that the digital portion of the chip is functional. Digital signals were clocked-in via SerDes, and read-out via the **D_OUT** port of the chip. Random binary patterns were applied to the input and cross-referenced with the received data, proving that the SerDes interface is functional. Nonetheless, the testing of the high voltage and NVM portion of the chip was inconclusive. Changes in the read-out current were observed after cell-erase operations but were not consistent with simulation results. It is believed that the level shifter circuits do not behave as per simulation, particularly, the variable power-supply SR-latch, whose body effect may be affected to point of bad operation. Additionally, there is a potential vulnerability during the high-voltage ramp-up phase, where gate-source voltages may be exceeded causing permanent damage to the chip.

3.5 Conclusion

A two-by-two floating gate array was designed and fabricated in a CMOS 55nm LPX using ESF3 Split Gate SuperFlash® [64] technology. Test results showed that NVM technology is capable of storing multi-bit values and was shown to be a promising technology for its use in Compute-in-Memory architectures. Test results allowed for both analog (VerilogAMS) and digital (Verilog) models to be developed and used in system-level simulation and full digital emulations in Xilinx FPGAs.

Using the digital and analog models of NVM arrays, a Compute-in-Memory (CiM) test-chip was designed and fabricated in CMOS 55nm LPX. The chip architecture is composed of a 156×512 NVM array as well as a Serializer/Deserializer (SerDes) interface that allows full access to rows & column programmability. The tested chip showed successful digital functionality, yet only showed minimum response in the analog circuits, leading to inconclusive experimental data.

After several technical discussions with Silicon Storage Technology, it was concluded that the level-shifting technique may have been the culprit, leading to permanent damage to internal devices upon high voltage ramp-up during programming and erase phases.

Chapter 4

Dynamic Random-Access Memory Primitives (DRAM)

4.1 Introduction

Thus far, Chapter 3 discussed non-volatile primitives for use in Compute-in-Memory (CiM) architectures. Nonetheless, it results natural to investigate more than one memory technology for use in silicon embedded processors. For the most part, when considering the term *embedded*, it means that the memory is integrated in the same die, or chip as the logic-computational unit. Similarly, it is desirable to attempt to embed CiM processors logic process, thus furthering the energy reduction by reducing off-chip memory access. Efforts in this work were supported partially by the DARPA UPSIDE [60] program, which was created to investigate unconventional ways to process high speed imagery, at extremely low energy.

Dynamic Random Access Memory (DRAM) has existed since the 1970s [67], and more recently has been made compatible with scaling logic processes [68], such as SOI 32nm, and FinFet 16nm [69]. Efforts for embedding DRAM stem from the increase in memory density compared to Static Random Access Memory (SRAM), typically by a factor of 8 [70]. Density makes this technology attractive for CiM processors; DRAMs provide natural cross-bar organizations which can be easily adapted to perform dot products in the charge domain at low energy expense and with massive parallel

capability.

Charge-based computing has been an attractive solution since its introduction with charged-coupled device (CCD) imagers in the seventies [71]. Such architectures have been adapted to CiMs that have been used for pattern recognition using the same underlying physics [72]. Other work has utilized the same concept in the charge-injection devices (CIDs) [14, 73, 74], which have also been used for similar pattern recognition tasks. However, these computing elements have not been implemented with feature sizes smaller than 180nm.

The principle of operation of DRAM consists of storing logic ones and zeros in small capacitors. Figure 4.1 shows a typical schematic of a DRAM bit-cell, comprised of an access transistor, and a capacitor. In order to operate a DRAM as memory, peripheral circuits are required to write, access, and refresh the leaking charge in the storage capacitor. In a specialized DRAM process, deep trench capacitors are used to achieve high capacitance, typically in the order of 0.1pF. However, in this work only standard features were available; thus, capacitors were created using MOSCAPS, resulting in relatively small capacitances, in the order of 0.05pF.

In this work, the CID concept is taken to a CMOS 55nm logic node; a mixed signal charge-based vector-vector multiplier (VMM) is explored, which computes directly on a common readout line of a dynamic random-access memory (DRAM).

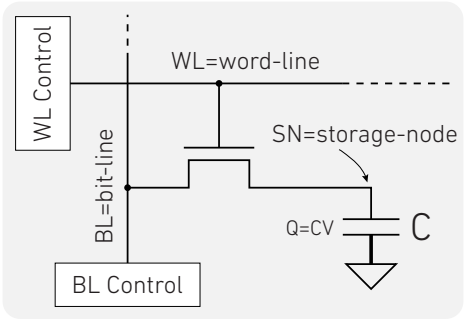


Figure 4.1 *Typical DRAM bit-cell, consisting of an access transistor and a capacitor (storage element).*

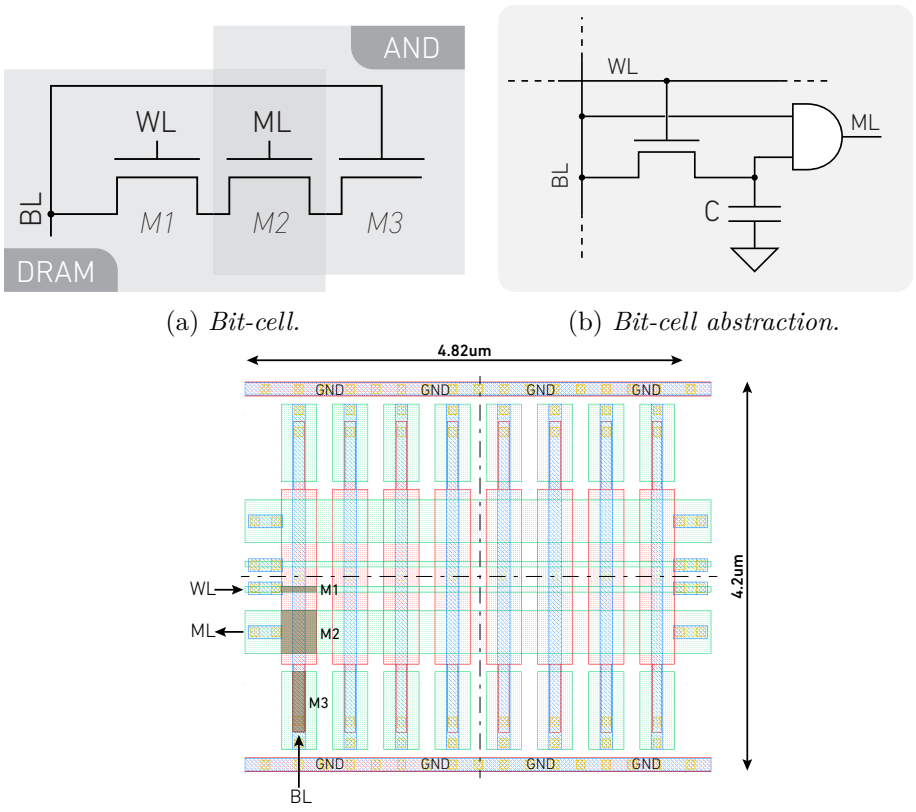
Low power consumption and high area density is achieved by storing local parameters in a DRAM computing crossbar. Section 4.2 shows the principle of operation of the array, Section 4.3 shows test-chip results, and Section 4.4 presents conclusions.

4.2 Pseudo-DRAM CiM Test-Chip

4.2.1 Cell Operation

The DRAM concept is translated to the charge-injection computational element, which is shown in Fig. 4.2a. From the figure, the access transistor (*M1*) is used for both reading/writing data from/to the storage element (*M2*), and the right-most transistor (*M3*) is used to perform an AND operation against the logic data on the bit-line. The operation is manifested as a change on voltage on the math-line, according to the work in [74]. Figure 4.2b shows an abstraction of the DRAM with the added computational element, and Figure 4.2c show the layout masks for: *Metal 1* (blue), *Poly* (green), *Diffusion* (red), and *Poly Contact* (Yellow). The 2×8 cell is used as a building block for larger arrays and allows the math-line and word-line metals to have frequent/low-resistivity connections to the polysilicon. The cell also contains plenty of bulk connections, which are delivered via the *GND* power-rails.

A logical one is represented in the bit-cell as the presence of negative charge $Q_{M2} = -(V_{ML} - V_{TH})C_{ox}$ under $M2$, while a logical zero is stored as the absence of charge. After writing to the DRAM, $M1$ is turned off and the trapped charge under $M2$ is used to perform a nondestructive logical AND operation against the data on the bit-line (BL). While the math-line (ML) is floating, if $x_j = 1$, the bit-line is pulsed high above the threshold of the half-transistor $M3$, causing charge to move from under $M2$ to $M3$ thus creating a tight capacitive coupling between BL and ML. As a result, ΔV_{BL} above the threshold of $M3$ will cause a proportional ΔV_{ML}. Conversely, if there is no charge stored under $M2$, a change in the BL voltage will only perturb ML due to parasitic coupling. Further details on cell operation and refresh can be found in [74].



(a) *Bit-cell.* (b) *Bit-cell abstraction.*

(c) *2×8 bit-cell layout.*

Figure 4.2 ***(a)*** *Pseudo-DRAM bit-cell,* ***(b)*** *its computational abstraction, and* ***(c)*** *a 2×8 layout which is used as the basic building block for larger arrays.*

4.2.2 Array Operation

The bit-cell in Fig. 4.2a can be arranged in a 2-D crossbar array, such as depicted in Fig. 3.3. The crossbar is constructed by connecting bit-lines vertically and word-lines/math-lines horizontally. The resulting array becomes addressable by random access, thus selecting a particular row (via the word-line decoder) allows the data be read-out and refreshed via bit-lines. Additionally, since all math-lines on a row are connected, any charge movement into the circuit via the compute transistor ($M3$) will be reflected as a voltage fluctuation on the respective math-line.

Figure 4.4 shows the equivalent compute-circuit, where the access transistors are turned off, therefore are not shown in the figure. The two remaining transistors in the bit-cell are in series and allow a 1-bit AND operation between inputs (X) and stored

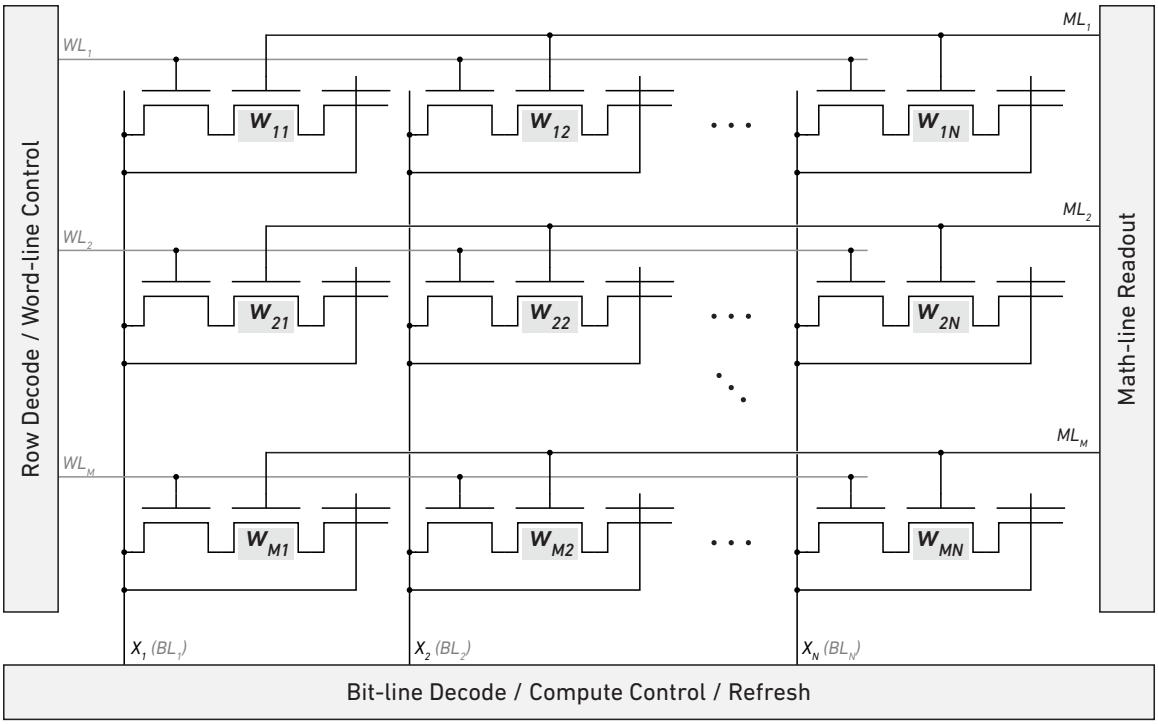


Figure 4.3 *Pseudo-DRAM crossbar showing common control lines and basic peripherals.*

weights (W). The effect of short-circuiting the math-lines creates a charge sharing effect over multiple parallel capacitors ($M2$), resulting in an analog averaging of the initial capacitor voltages. Using basic physical principles (and detailed in [74]), it is easily shown that the change in math-line voltage is given by Eq. 4.1, shown below:

$$\Delta V_i = \frac{\alpha}{N} \Delta V_{BL} \sum_{j=0}^{N-1} x_j w_{ij}, \tag{4.1}$$

where α is a constant that depends on both oxide and junction capacitances from the bit-cell in Fig. 4.2a, and ΔV_m is the change in the bit-line voltage above the threshold of $M3$. If the coefficient outside the summation term is removed, then the operation can be abstracted as a simple dot product between $X = (x_0, x_1, \ldots, x_{N-1})$ and $W = (w_0, w_1, \ldots, w_{N-1})$, shown below (Eq. 4.3):

$$Z_i = \sum_{j=0}^{N-1} x_j w_{ij}, \tag{4.2}$$

$$Z_i = X \cdot W_j^T. \tag{4.3}$$

Moreover, if the weight matrix is written as $W = (W_0^T, W_1^T, \ldots, W_{M-1}^T)$, then the

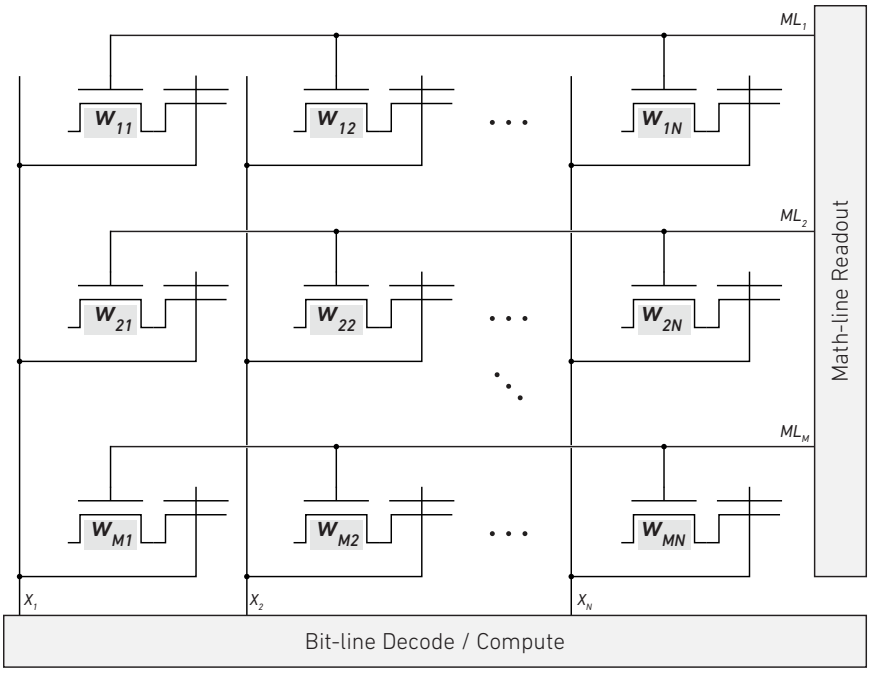


Figure 4.4 *Pseudo-DRAM crossbar equivalent circuit for computation.*

output, $Z = (Z_0, Z_1, \ldots, Z_{M-1})^T$ can be written in the form of a vector matrix multiplication, $Z = XW$.

Output Precision

Recalling from Eq. 4.3, weights and inputs (w_{ij}, x_j,) are 1-bit parameters, hence the precision of Z_i is $P(Z_i) = \lceil \log_2(N) \rceil$, where N is the number of columns. It is possible to achieve higher parameter and input precisions via stochastic or binary weighted encoding, which is explained in detail in Chapter 2.

4.2.3 Core Architecture

The core architecture of the test-chip is depicted in Fig. 4.5. The design was created in a bottom-up full-custom approach, where the basic building blocks were created, optimized, and later tiled together to create the system shown in the figure. The system consists of the following building blocks:

- A 156×512 pseudo-DRAM crossbar. This array is split into 8-column slices,

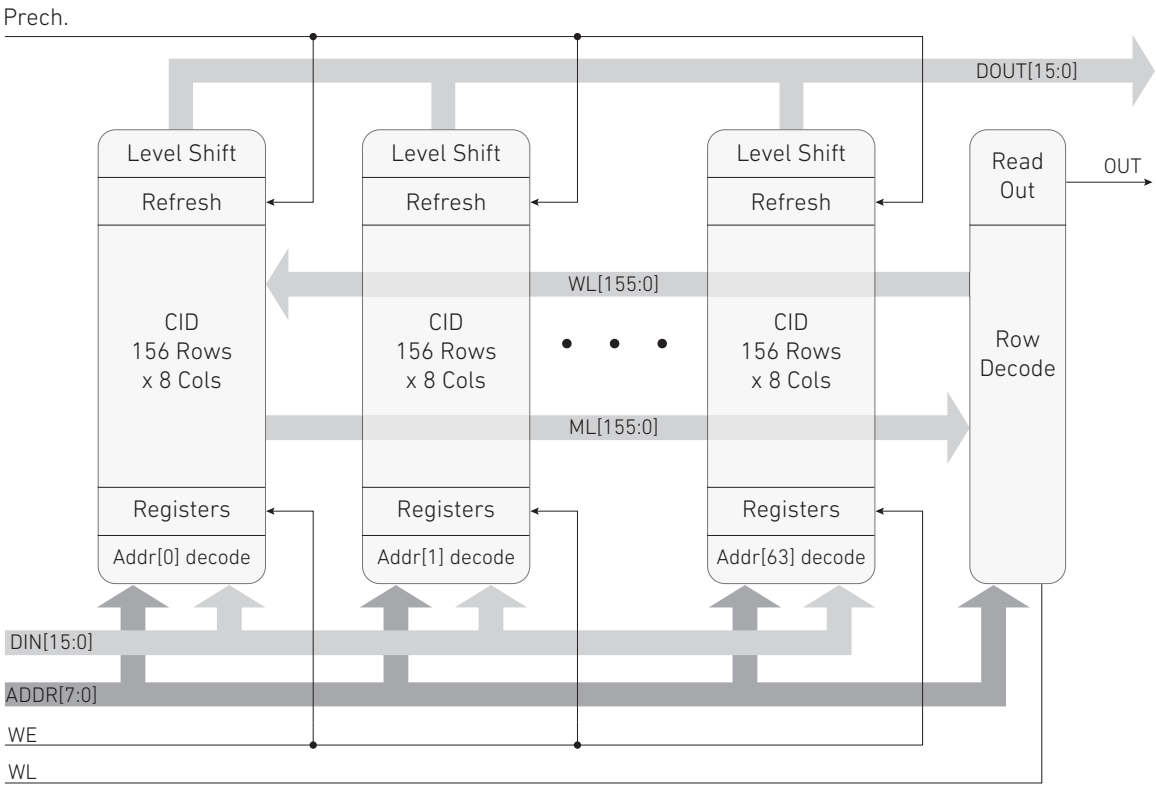


Figure 4.5 *Pseudo-DRAM CiM micro-architecture shown its chip-level interface.*

each requiring 16-bit configuration parameters. Moving down in hierarchy, the basic building block was a tiled 2×8 charge injection device bit-cell.

- A latch-based input data-bus for: (i) writing to the array and (ii) selecting rows to output after a computation. A column-parallel address decoder allows slices of 8-columns to be addressed individually, followed by registers that are used to buffer input data while the rest array is been written.

- A tri-state output bus, which based on the input address allows refreshed data to be read-out of the chip.

- A math-line read-out circuit, which consists of a voltage buffer and an analog multiplexer. Any of the 156 analog voltages can be routed to the analog output of the chip.

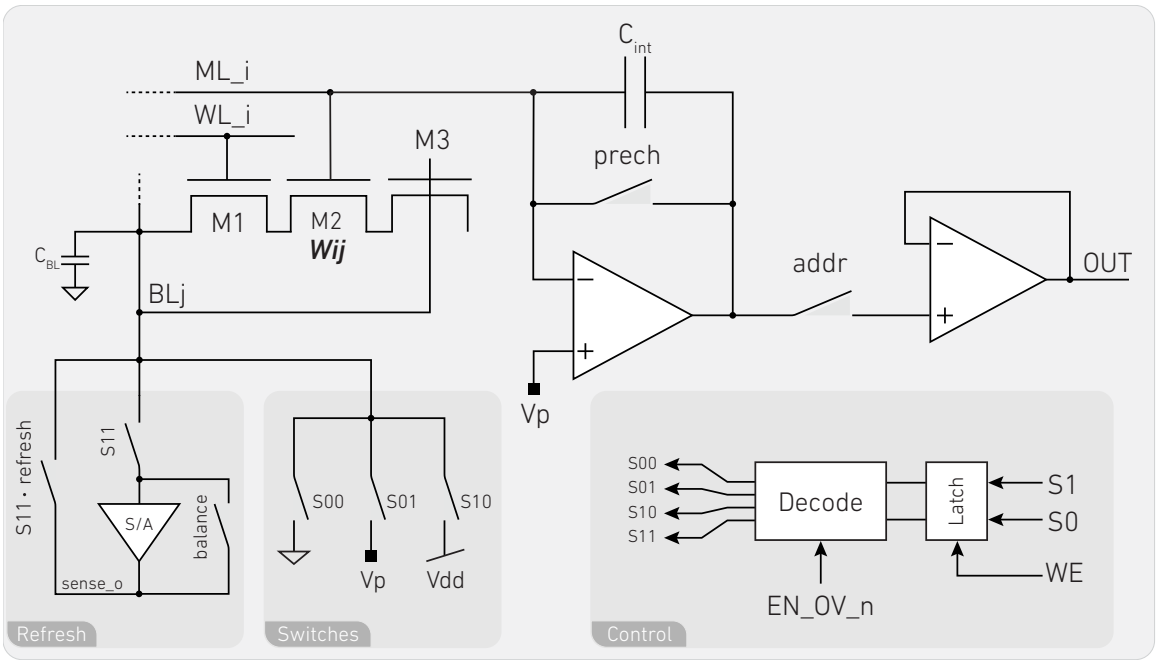


Figure 4.6 *Full schematic diagram of pseudo-DRAM bit-cell and its peripheral circuits.*

4.2.4 Analog Peripheral Circuits

Figure 4.6 shows a full schematic diagram of the pseudo-DRAM bit-cell and its peripheral circuits. Only one bit-cell is shown and assumed to be fully connected to the rest of the elements in a larger array. The main blocks in the design are listed below:

- A Charge-Injection-Device (CID) bit-cell, consisting of a DRAM storage element and a computational element.
- Read-out circuits, which consist of a switch-capacitor amplifier, and a voltage buffer.
- A sense-amplifier, used to sense and refresh data in the DRAM array.
- A series of switches, which allow fine-grained control over the bit-line voltage.
- Simple digital control, consisting of a pair of latches and a decoder, which in turn controls the state of the switches described in the previous item.

Table 4.1 ***(a)*** *Data encoding for control switches in CID bit-cell. The EN_OV_n must be high during a write/compute sequence. When the former signal is low, it forces the S*00 *switch to close, thus overriding the state of the front-end latches.* ***(b)*** *State of switches for writing or computing with different logic levels in the input.*

(a)

S1	S0	S00	S01	S10	S11
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(b)

Operation	Logic 0	Logic 1
Write	S01	S00
Compute	S00	S10

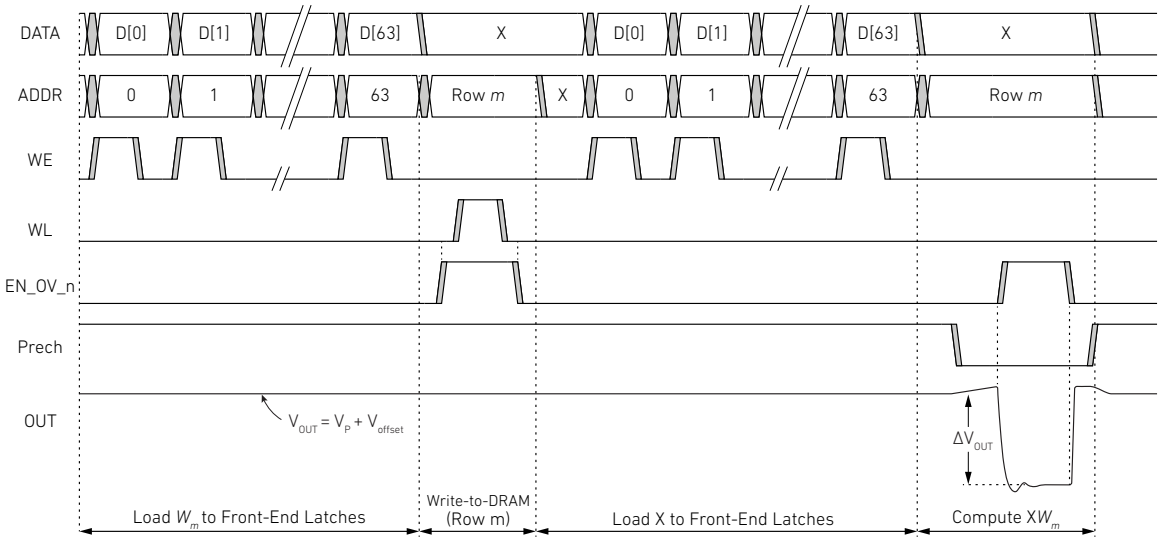


Figure 4.7 *Timing diagram for writing and computing in the CID array.*

Write Operation

Before computing, the weight matrix, W, must be written to the array. The write operations takes place by setting bits *S1* and *S0* at the front-end of the column (using the encoding scheme in Tables 4.1a & 4.1b), and later pulsing the appropriate word-line. Initially, from Fig. 4.6, the *prech* switch remains closed, therefore clamping the gate of $M2$ to $\sim Vp$, which is set to be approximately $Vdd/2$. Next, the front-end latches are loaded with bits *S1* and *S0* by spanning the address range $0-63$. Finally, the address is set to target the desired row, followed by a pulse on the word-line (WL).

During a write phase, if the bit-line is driven to Vp, then a logic *0* is written to

the cell, thus no (or little) charge will be stored on the gate of $M2$. Conversely, if the switches drive the bit-line to ground, a total charge of $Q = C_{ox}(Vp - V_{th})$ will be stored on the gate of $M2$, which signifies a logic *1*; the presence of charge will create a tight capacitive coupling between the bit-line and math-line. The full write sequence is illustrated in the left portion of the timing diagram of Fig. 4.7, where all latches are written in batches of 16 (which is the bus width). During the process of writing data to the front-end latches, it is desirable to keep the bit-line at ground, which is why the the EN_OV_n signal is incorporated; while low, the $S00$ switch is closed, forcing the bit-line to ground. Later during the write-to-DRAM phase, the EN_OV_n signal is set to *1* and the bit-line will shift to the value set by signals *S1* and *S0*. Noteworthy, all signals shown in the timing diagram are wired to the periphery of the chip and are assumed to be controlled via an external device.

Compute Operation

Once all parameter data, W, has been written to the array, the front-end registers must be loaded with the the input, X, in preparation for a compute sequence. The key to computing XW lies in the precharge switch-capacitor amplifier. During the write-phase, it clamps the math-line to $Vp + V_{offset}$ by holding the *prech* switch closed. Prior to compute, the switch is opened and subsequently the EN_OV_n signal is set high, causing the bit-line to either stay at ground, or pulse to *Vdd* or *Vp*. Since the amplifier clamps the math-line, all excess charge in the circuit will be forced to integrate into the feedback capacitor, C_{int}. The output of the amplifier will decrease below its resting potential by an amount proportional to $\sum_i x_i w_{ij}$. Later on a row-by-row basis, the output buffer is used to select which analog signal to read. The compute-sequence is illustrated on the right-hand side of the timing diagram of Fig. 4.7. The diagram shows some detail of the behavior of the analog output, which increases slowly after the *prech* switch has opened. This is caused by leakage in the

DRAM that effectively is integrated by the amplifier. Likewise, after the EN_OV_n pulses high, the slew-rate of the amplifier limits the output voltage, which finally settles to a stable value after some oscillation.

The switch-capacitor amplifier was built using a two-stage operational transconductance amplifier (OTA), and was stabilized with a *Miller* capacitor in order to handle a 1MHz bandwidth with a good damping factor. The read-out buffer is a two-stage operational amplifier (OPAMP), with enough drive strength to drive a 20pF pad at 200KHz.

4.2.4.1 Read Operation

In DRAM, a read operation consists of opening a row (by selecting the appropriate word-line) and causing the charge in the storage-node (SN) to be shared with the

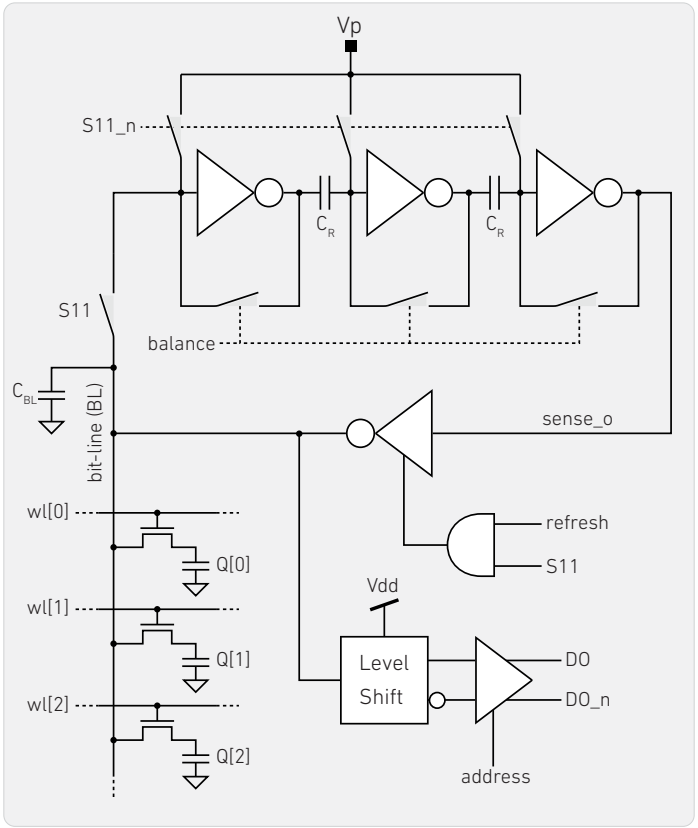


Figure 4.8 *Schematic for the DRAM refresh circuit. During a read operation, a sense amplifier is used to detect the charge polarity in the storage capacitor.*

parasitic capacitance of the bit-line. A carefully designed sense-amplifier detects a voltage change, sends the data to the read-bus, and writes it back to the DRAM cell. Figure 4.8 shows the refresh scheme used in this work, whose design considerations are listed below:

- **No access to deep-trench capacitors:** Since access to deep-trench devices was not an option, the array was driven to be as small as possible, while still having the capability to sense and refresh charge. Effectively, the size of the bit-cell is large, which pushes the design to save area in other ways.

- **Single-ended storage:** in this work, memory density was considered the prime driving factor. Normally DRAM arrays utilize differential storage, which relaxes the constrains on the sense amplifier and increases its reliability. In this work, single ended storage was used, thus increasing memory density by a factor of two. Consequently, this creates a particular challenge when it comes to refresh, thus the voltage difference on the bit-line was simulated to be only a few millivolts.

- **Fabrication mismatch:** mismatch in traditional sense amplification schemes (see [75]) resulted larger than the target sense voltage. This was verified by using Montecarlo simulations, and effectively drove the design process towards a switch-capacitor differential sense-amplifier.

Henceforth, the sense amplifier shown in Fig. 4.8 was designed to be insensitive to process mismatch, and capable of refreshing data on single ended storage. The *S11* switch is used to connect the bit-line to the input of the sense amplifier. Under these conditions, the inverters are short-circuited (input-to-output) via the *balance* switches. This causes all inverters to settle at their maximum gain point (also known as *trip point*), charging the parasitic capacitance of the bit-line to approximately $Vp/2$. Decoupling capacitors are used to cascade inverters, thus achieving a total gain of G^3, where G is the gain of a single device. Upon opening of the *balance* switches,

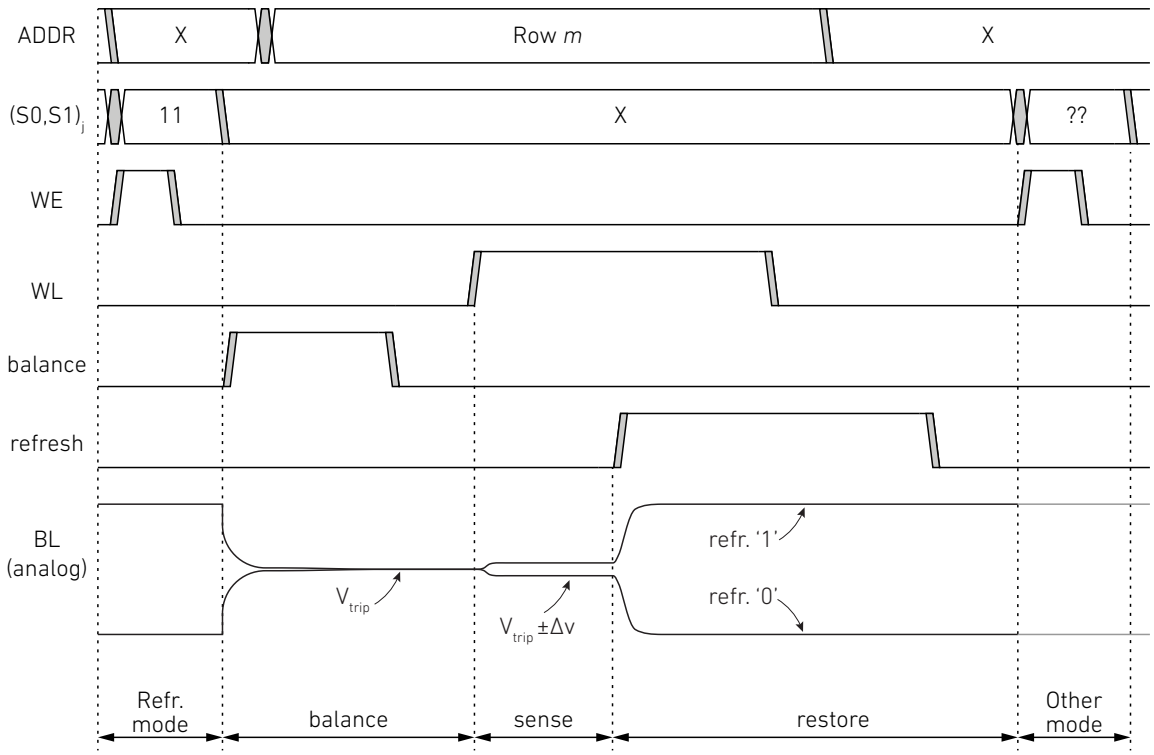


Figure 4.9 *Timing diagram for the refresh operation.*

the inverters are ready to sense charge; the word-line is pulsed high, charge from the DRAM is shared with the parasitic capacitance of the bit-line, and a small voltage change is sensed and amplified by the chain of inverters. Once the last inverter has converged, the *refresh* signal is pulsed high, driving the bit-line to the refreshed-value. The whole sequence is captured in the timing diagram in Fig. 4.9. The bottom-most signal is an abstraction of the bit-line voltage over time. Finally, the word-line is pulsed low, charge is once again trapped in the storage-node, and the sense amplifier can be used to scan and refresh the rest of the array. Additionally, the sensed data is level-shifted back to the I/O voltage domain (1.2v in this test-chip) and connected to a tri-state bus, which is enabled with the address. The address range for targeting rows is set to be from $100 - 255$.

Noise Analysis

Capacitive circuits are subject to noise; one approach to analyzing this effect is to consider a parallel resistor and its equivalent Johnson-Nyquist (thermal) noise. An equivalent noise circuit is shown in Fig. 4.10, where the variance (or power) is given by $\overline{v_r^2} = 4k_BTR$, and the real-mean-square (RMS) value is given by $v_r = \sqrt{4k_BTR\Delta f}$. k_B is the Boltzmann constant, T is the temperature in K, and Δf is the bandwidth over which the noise is considered. The noise current will integrate in the capacitor (also as white noise), and following the approach in [76], it is easily shown that $v_c = \sqrt{K_BT/C}$. The result shows that the noise does not depend on the series resistor, thus the bandwidth of the circuit depends on RC; as R increases, the noise increases, but the bandwidth decreases by the same amount.

Given that the charge-injection compute-unit relies on charge sharing in capacitors, it is important to analyze the noise and derive an upper bound on the output precision. Essentially, if a compute operation is carried out in an effective voltage drop of ΔV, then the thermal noise in the circuit should be less than $\Delta V/(2^B)$, where B is the number of bits of the desired output. Capacitor sizes in this work are in the order of 0.5fF, yielding a noise amplitude of approximately 2.84mv. This means that in order to achieve 8-bits, a voltage swing of 0.72v is needed. The total number of columns in the array was set to twice of the minimum to achieve 8-bits and was chosen to compensate for unforeseen non-linearities. Additionally, if two columns per bit are used, then only a 0.512v voltage swing is needed in order to achieve 8-bits.

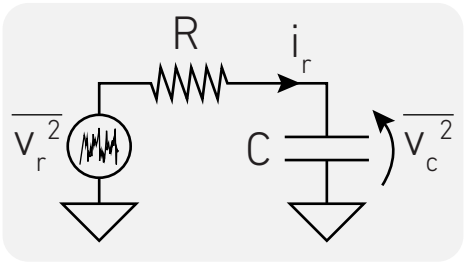


Figure 4.10 *Thermal noise in capacitors.*

On a side note, Montecarlo simulations showed that the effective number of bits is limited by process mismatch, not by thermal noise. By simulation, the system can obtain around 6-bits of precision.

4.3 Results

The computing architecture described in the previous section was fabricated in a CMOS 55nm logic process, with a power supply of 1.2v. The system comprises a 156×512 CID array, which is a full-custom design and was simulated to operate at an internal supply of 0.6v–0.9v. Figure 4.11 show the top level layout of the final design and its internal blocks, and Table D.1 shows the top level pinout of the chip. The effective size of the block is 500×700μm and was placed inside three power rings and integrated in a larger design. A micrograph of the chip is shown in Fig. 4.12, where the larger design is also shown. The chip was bonded on a 145 pin CGP package, which was mounted on a custom printed circuit board.

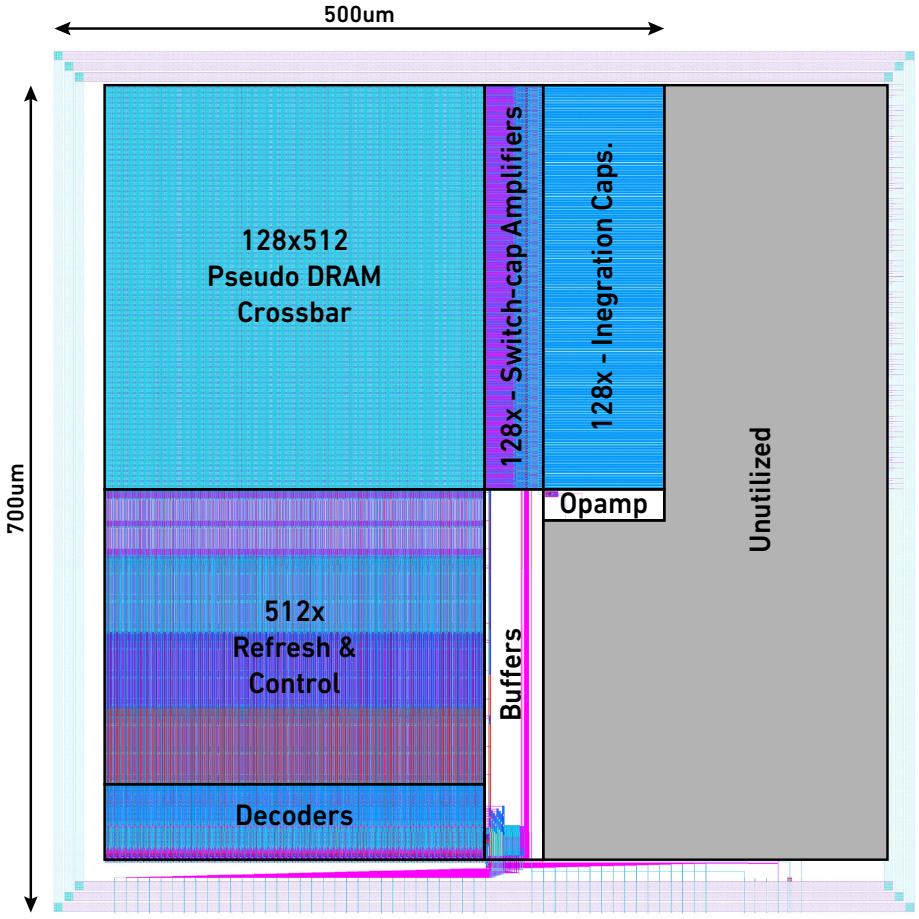


Figure 4.11 *Top level layout of the pseudo-DRAM array and its internal blocks.*

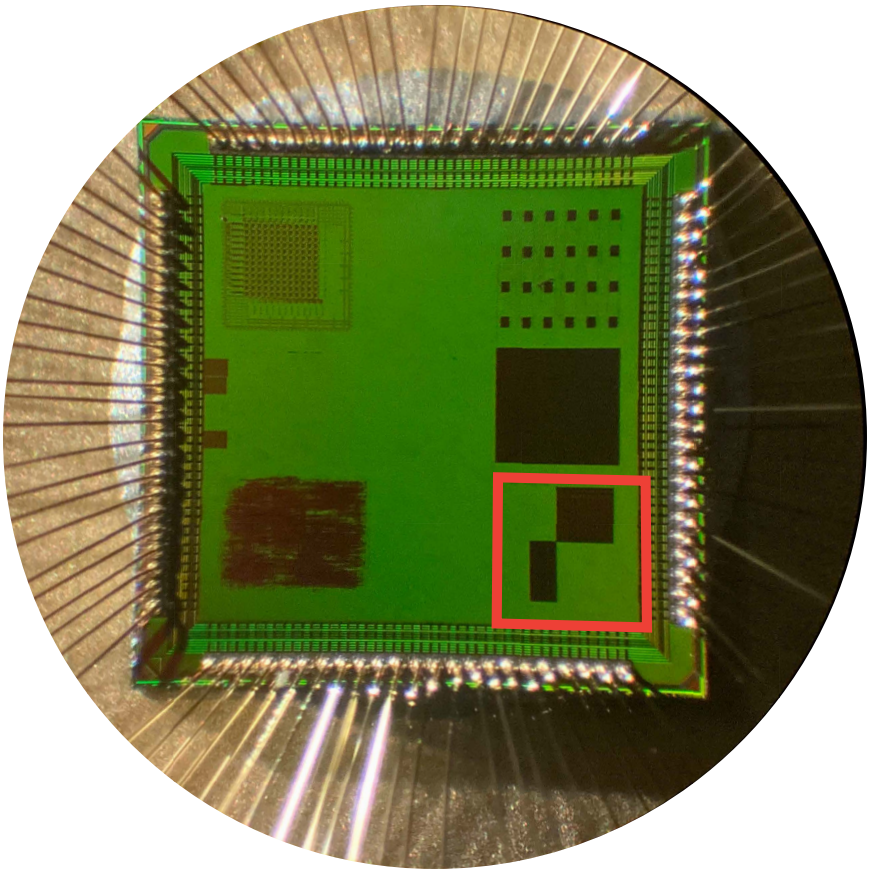


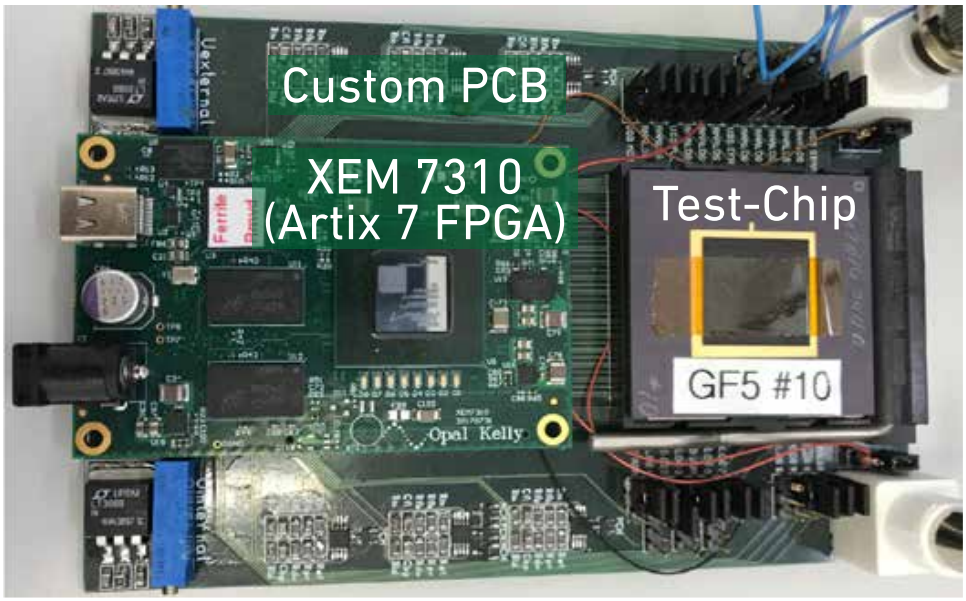
Figure 4.12 *Micrograph of fabricated test-chip, which is part of a larger design.*

4.3.1 Experimental Setup

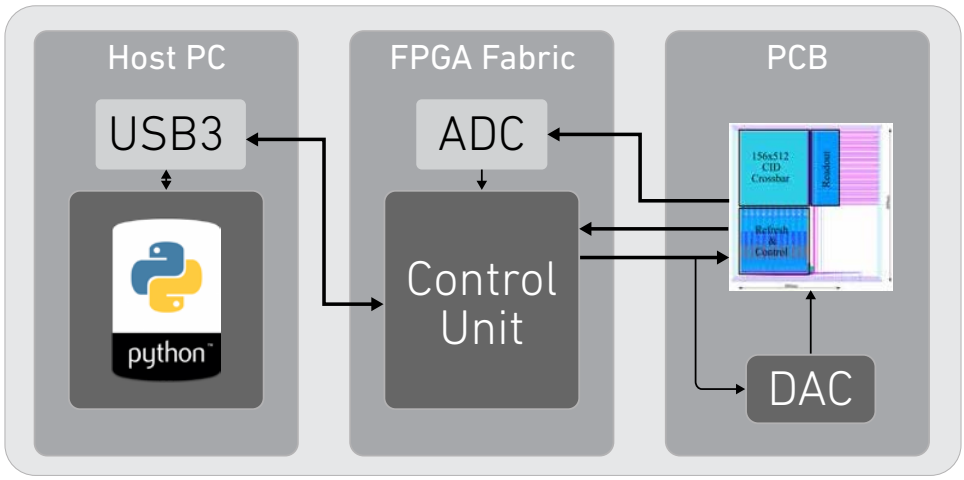
The packaged test-chip was mounted on a custom printed circuit board (PCB), which in turn attaches to the FMC connector of a Xilinx XEM-7310 FPGA. A picture of the setup is shown in Fig. 4.13a. The custom PCB comprises on-board regulators, D/A converters, and breakout pins for oscilloscope probing. A block diagram of the same experimental setup is shown in Fig. 4.13b. The setup has three main components:

- **Host PC:** Ultimately, the chip is controlled via a host PC (or Mac), which communicates with an FPGA via USB3. All software development for testing is done in python via the Opalkelly SDK. The end-user is able to monitor all mapped internal registers in the FPGA and I/O pins of the chip though *wires* or *pipes*, providing up to 300MB/s of data throughput to/from the FPGA.

- **FPGA:** An Opalkelly XEM-7310 Artix-7 A200 FPGA, which is the bridge between the test-chip and the host PC. The Artix-7 additionally has an on-chip A/D converter, which is used as the main conversion device in the work presented in this chapter. The FPGA fabric presents a sophisticated controller, capable of providing high throughput between the chip and the host PC as well as providing precise, reconfigurable timing of all control signals that are



(a)



(b)

Figure 4.13 ***(a)*** *Photograph of the experimental setup.* ***(b)*** *Block diagram of the experimental setup.*

shown in Figs. 4.7 & 4.9. Additionally, the control unit provides a master serial-peripheral-interface (SPI) that controls the D/A converters on the PCB.

- **PCB:** The printed circuit board has four main components: (i) FMC connectors that connect directly to the Opalkelly board, (ii) on-board voltage regulators, which are used to set the I/O voltage on the FPGA pins and supply power to the test-chip, (iii) digital to analog converters (DACs), which are used to generate all biases and low power supplies for the chip, and (iv) a pin grid array (PGA144) socket will is used to connect the chip.

4.3.2 Preliminary Results

The experimental setup described in the previous section was carried out using the following settings, and results are partially extracted from [77]:

- Bias Precharge Opamp: 0.42[v].
- Bias Output Opamp: 0.53[v].
- Precharge voltage: 0.8[v].
- Compute ΔV=0.8[v].
- ADC: 12-bits, 700Ks/s, triggered internally on the compute signal. Buffer size acquired: 4096 (5.8[ms]).

The first step in testing was to create a spatial density encoding in both X, while holding all CID parameter weights at logic *1*. By initial inspection using an oscilloscope, it was verified that the buffered output math-line change in voltage was indeed proportional to X. Figure 4.14 shows the output ΔV for different spatial encoding of X, ranging from 0 to 5/8.

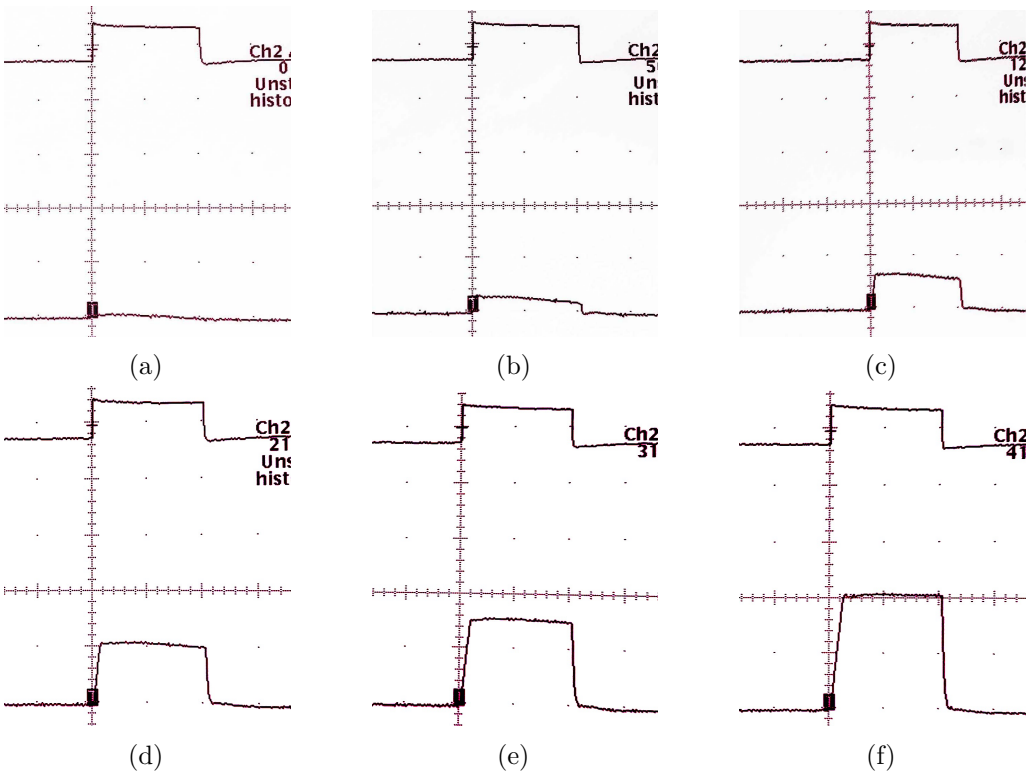


Figure 4.14 *Buffered math-line pulses during a single row readout. The horizontal axis is is set to 2.5μs/div, and the vertical axis is set to 200mV/div. The top traces correspond to EN_OV, and bottom traces correspond to* $\delta V = Vp - V_{out_m}$*, where m is row 150 for this particular graph. Pulse density on the input vector X was set to:* ***(a)*** 0, ***(b)*** 1/8, ***(c)*** 2/8, ***(d)*** 3/8, ***(e)*** 4/8 , *and* ***(f)*** 5/8. *A pulse density of* 1/2 *equates to 256 ones and 256 zeros on the X input.*

4.3.3 Results using Xilinx A/D

Since oscilloscope inspection was successful, the on-chip Artix-7 A/D converter was used to sample the buffered math-line. Conceptually, the output of the A/D was wired to a synchronous FIFO, which in turn was used to stream data back to the host PC by using the Opalkelly interface. The A/D converter was sampled at 700Ks/s and buffered for 5.8[ms] per conversion. A time-domain plot of captured waveforms is shown in Fig. 4.15a. The encoding used is the same as in the previous section, where all weights were set to logic ones, and X was an incremental spatial PWM signal.

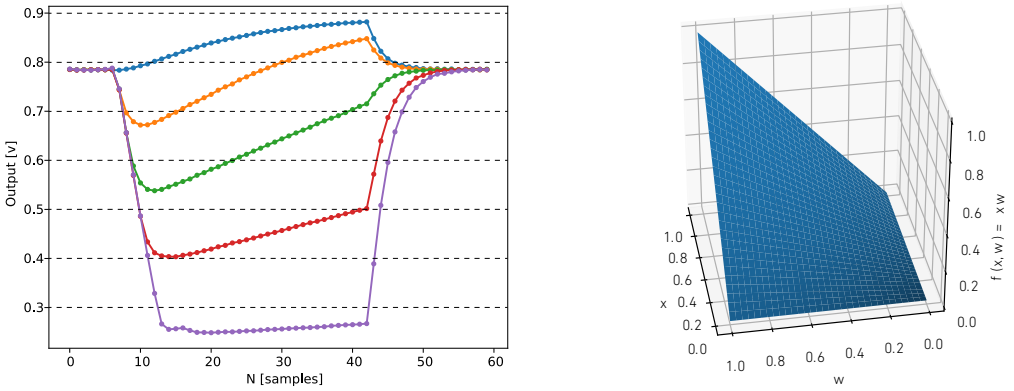
From Fig. 4.15a, a few interesting aspects are noted:

- **Slew rate:** the falling edge of the output is slew-rate limited. Increasing the bias

current of the OTA and OPAMP improved the slope but introduced ringing in the output. Slew poses an interesting challenge in data collection, thus sampling the ADC too soon as well as too late could result in garbage data. Data in this report was collected at the 13th sample, where it is insured that all outputs have converged regardless of the value. This selection was based on empirical results.

- **Leakage:** It can be seen from the figure that after the output reaches a minimum, it starts increasing gradually with a somewhat linear slope. This effect was verified via simulation and is attributed to leakage in the DRAM transistor. Gate-oxide tunneling current is the main leakage source but there is an additional effect that is believed to be related to sub-threshold between the bit-lines and the storage transistor's channel; a logic one (for x) during the compute-cycle will cause negative charge from $M2$ to move towards the bit-line. Consequently, at higher input compute-densities, the leakage effect is reduced in the output.

- **Parasitic coupling:** Finally, it was observed that parasitic coupling between the bit-line and math-line caused a slight output change even when the weights were programmed to be zero. The coupling is attributed to the overlap capacitance between the gates and drain/source of $M2$ and $M3$. This effect can be considered



(a) (b)

Figure 4.15 ***(a)*** *Time domain output captured with Xilinx on-chip A/D converter.* ***(b)*** *One quadrant, ideal multiplication, $f(x, y) = x\,w$, 3d-mesh plot used for error extraction and comparison.*

to be a systematic error, thus it is very predictable and easily eliminated; the key is to compute $X(W_{zeros})$ (dot product between the input and a zero-weighted row), and later subtract the result to the desired output. This process yields $(XW)_{corrected} = XW - X(W_{zeros})$.

Figure 4.15b shows a surface plot of an ideal 1-quadrant multiplication (1/4 of a saddle). This surface is considered the ground truth for the 1-quadrant product and will be used to assess the quality of the results.

Stochastic Modality

The CID crossbar can be used to compute multiplication in two main modalities: (i) stochastic and (ii) deterministic. In this section, stochastic encoding of weights and inputs is explored.

The key idea is to utilize space (as opposed to time), to represent data. Recalling from Chapter 2, let us consider two random binary streams, $x = (x_0, x_1, \ldots, x_{L-1})$ and $y = (y_0, y_1, \ldots, y_{L-1})$, where x_i and y_i are Bernoulli random variables with probabilities p_x and p_y. When both bit-streams are multiplied elementwise ($z_i = x_i\, y_i$), the averaged output is:

$$z = \frac{1}{L} \sum_{i=0}^{L-1} z_i, \tag{4.4}$$

and has a Binomial distribution. Furthermore for a large integration count, the distribution tends to a Normal with mean $p_z = p_x\, p_y$, and variance $\sigma_z^2 = p_z(1 - p_z)/L$. The pseudo-DRAM crossbar is suitable to perform this kind of averaging, and is particular interest because it leverages parallelism and charge-computing to perform the averaging in Eq. 4.4.

True Random Encoding

Using the test-bench described in Fig 4.13, a Python script was created to encode digital inputs in the stochastic domain. The encoding scheme that was used is depicted

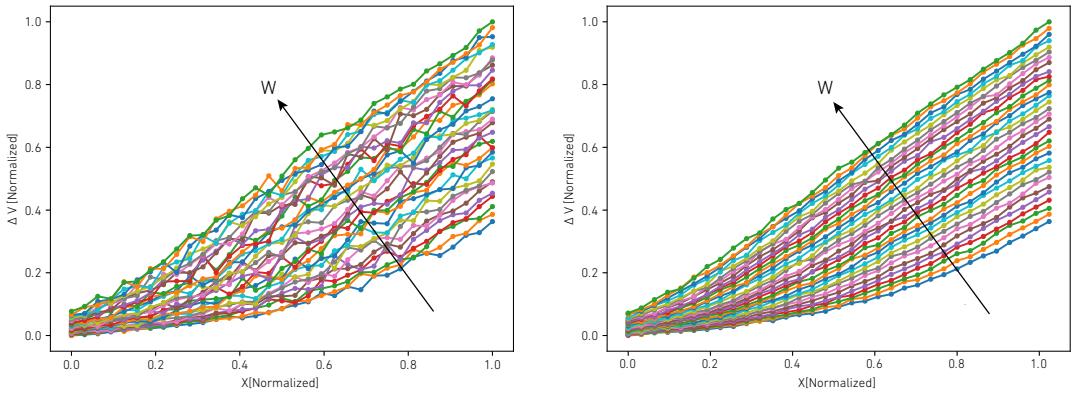
in Fig. 4.16a, where a digital value was compared to a true random variable with a probability of 0.5. Parameter data was loaded in the CID array and used to compute multiplications. Figure 4.17a shows a plot of $p_x\, p_w$ when inputs are swept over their valid range, which is from 0–512. Clearly, the stochastic nature of the encoding causes variability in the output. The offset at zero-valued input is attributed to leakage effects in the array, explained in the previous section.

Pseudo-Random Encoding

In order to reduce true random effects in the inputs, linear feedback shift registers (LFSRs) were used for encoding, such as depicted in Fig. 4.16b. The resulting pulse-density modulation yields an impulse-like autocorrelation function, implying that shifted versions of the data have low, or controlled correlation, making them suitable for stochastic computing. The distribution of the binary stream is uniform, yet if

(a) (b)

Figure 4.16 ***(a)*** *True random and* ***(b)*** *pseudo-random stochastic encoding scheme.*



(a) (b)

Figure 4.17 *One quadrant, stochastic sweep for X and W using* ***(a)*** *random encoding and* ***(b)*** *pseudo-random encoding.*

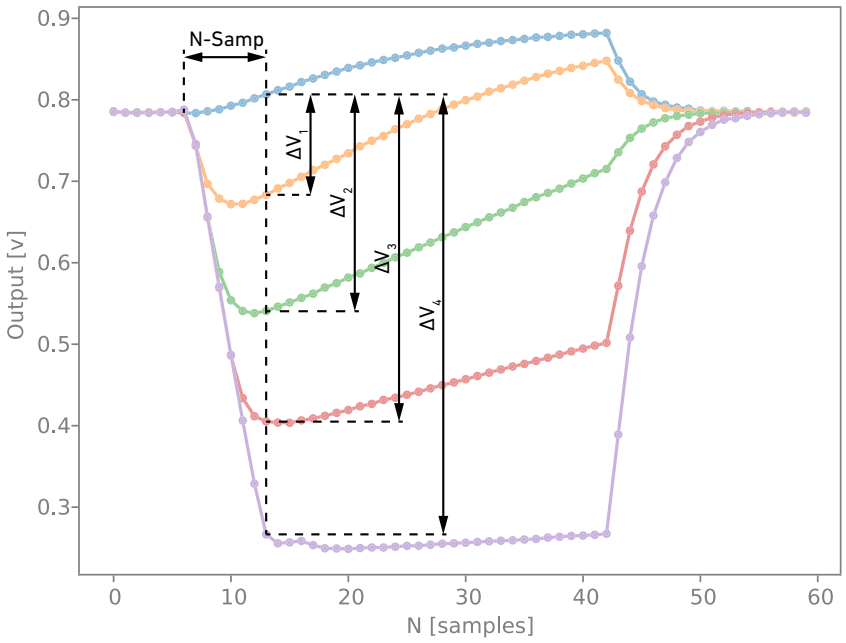


Figure 4.18 *Sampling scheme to eliminate zero-offset.*

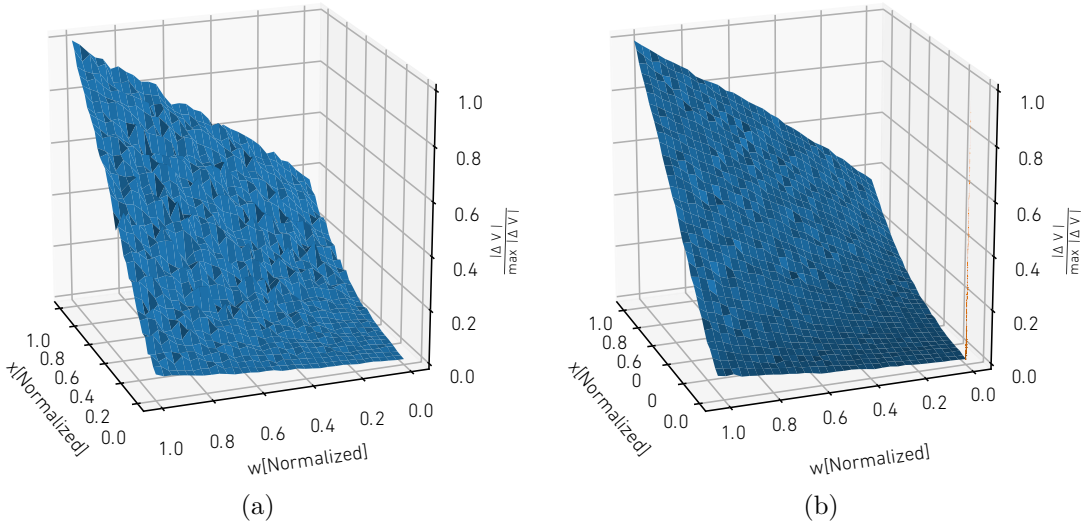


Figure 4.19 *Stochastic multiplication* $f(x, w) = x\,w$ *plotted on a 3D mesh using* ***(a)*** *random encoding and* ***(b)*** *pseudo-random encoding.*

an encoded value is integrated for exactly one period of the LFSR, then the exact encoded value is achieved. Figure 4.17b shows test results for swept inputs and weights over the entire space. Clearly, for a particular value of input X, the variance of the output is smaller than that of the true random encoding. Zero offset is still present but can be eliminating by calculating the ΔV of the output with respect to a leaked

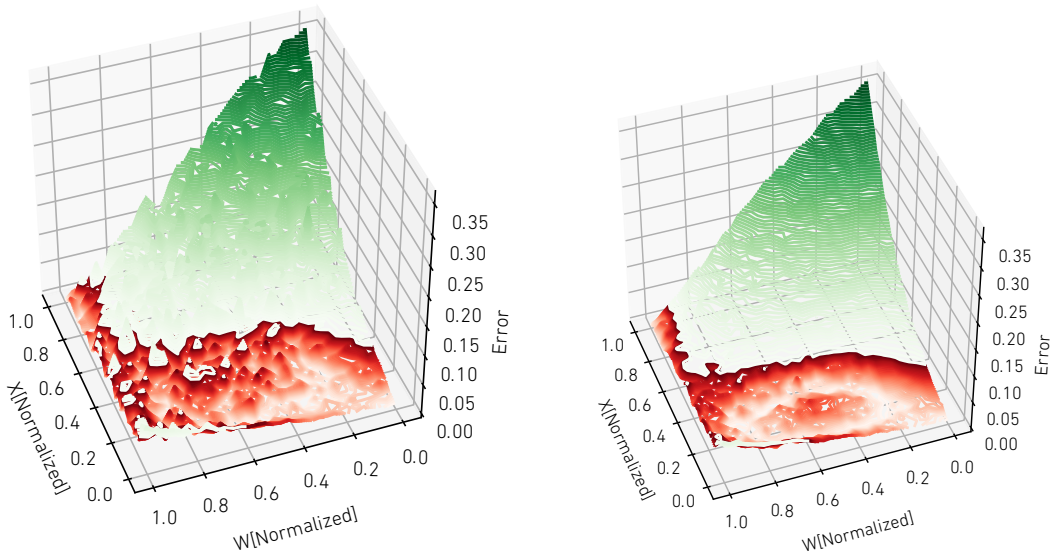


Figure 4.20 *Error function between ideal multiplication when using* ***(a)*** *random encoding and* ***(b)*** *pseudo-random encoding.*

zero-weighted output. Such a procedure is illustrated graphically in Fig. 4.18.

The random encoding schemes in this section were used to create quarter saddle plots. Results for the true random encoding are shown in Fig. 4.19a and results from the pseudo-random encoding using LFSRs is shown in Fig 4.19b. Error plots for encoding methods are shown in Figs 4.20a & 4.20b, respectively. The red areas in the figure shown the regions for which the output result has a 5-bit accuracy, or better. Without any compensation techniques (which will be discussed later), a 5-bit result may be achieved for inputs that are smaller than 0.5 (or $512/2 = 256$).

Deterministic

Within the scope of this section, deterministic computing modality will be considered as spatial pulse-width modulations (PWMs). Figure 4.21a shows the swept input-parameter space while using PWM, which is accomplished by comparing a ramp with the input/parameter to be encoded. The family of curves was created by storing constant spatial pulse densities in the array, and subsequently pulsing an all-ones input

vector ($x'_i = 1\,\forall i$) over x iterations. For each iteration, the output was accumulated off-line, yielding $z = x\,w$. The process was repeated over all possible w values, and data was post-processed in Python.

The zero-offset was removed by sampling the ADC at the 13th sample (considering fs=700KHz) from the *start_compute*, and later considering the output as the voltage difference between a zero-valued output that has been leaking for the same amount of time. This process is illustrated graphically in Fig. 4.18. Additionally, there is a systematic gain error due to parasitic coupling between the gate-drain overlap capacitances of $M2$ and $M3$. This was easily eliminated by subtracting the zero-weighted output to the whole family of curves. The former can be computed in parallel and subsequently subtracted either off-line or in the FPGA. Results after zero-offset compensation and gain-error correction are shown in Fig. 4.21b, and the corresponding error plot is shown in Figs. 4.22b. The red region represents the computing space for which the output precision is 6-bits or more.

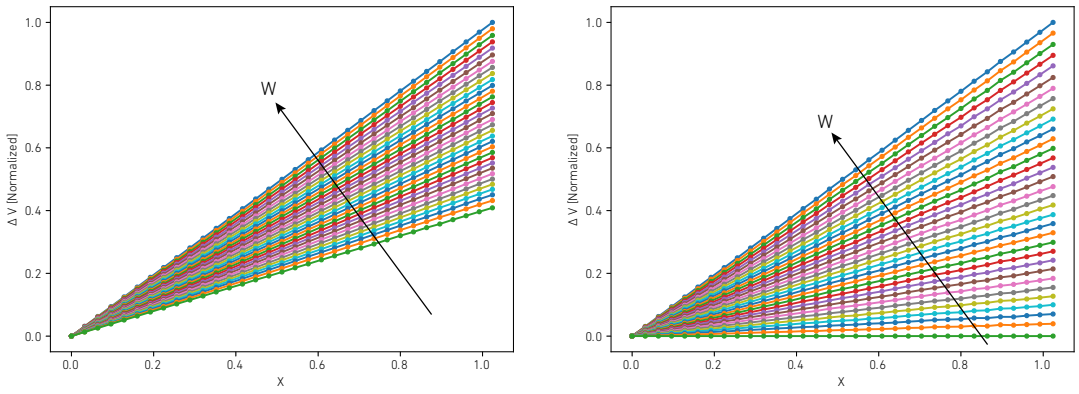
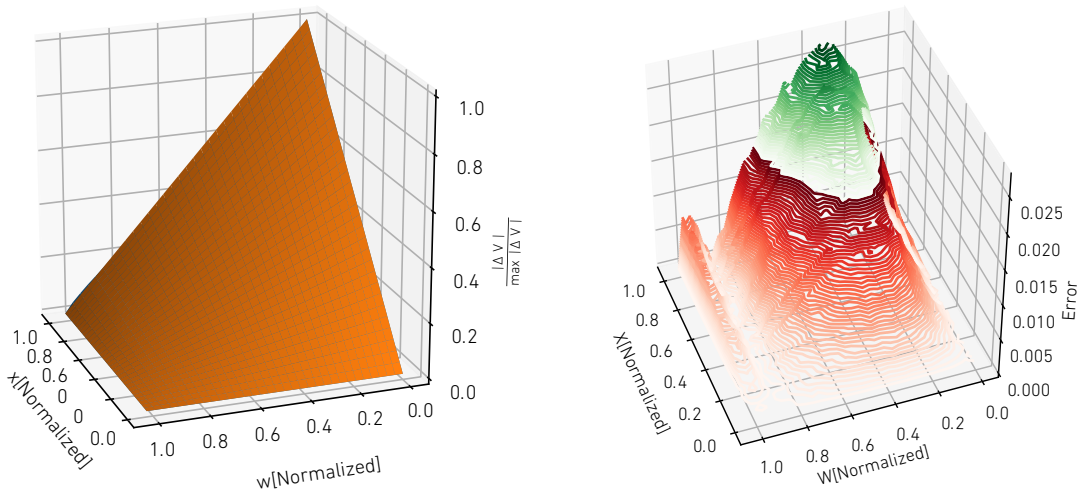


Figure 4.21 *One quadrant sweep for X and W with a spatial pulse-width-modulation encoding* ***(a)*** *without systematic error compensation and* ***(b)*** *with systematic error compensation.*



(a) (b)

Figure 4.22 ***(a)*** *Deterministic multiplication and* ***(b)*** *error when using spatial PWM encoding of X and W.*

Core Efficiency

Core efficiency was calculated based on full-system simulations by measuring the current in the main power supply. The design specs for the tested chip are shown in Table 4.2, and include only the mixed-signal core. The compute speed is limited by the slew-rate and bandwidth of the internal precharge amplifier; at a 10MHz bandwidth, compute speed is adopted as 1MHz (∼BW/10).

The energy for a single compute over one row is calculated considering both static and dynamic components. The static power comes from the OTA's tail current, and dynamic power comes from charging the total math-line (ML) capacitance, $C_{ML} = \sum_i C_{bit} + C_{wire}$, as well as the integration capacitance, C_{int}, in the feedback loop of the OTA. In order to achieve unity gain in the OTA, C_{int} was set to be equal to C_{ML}. The dynamic energy is given by $E = 2 \times C_{int} \times \Delta V^2$, where ΔV is considered in the worst-case-scenario, i.e. the maximum voltage change on the output of the OTA.

The dynamic average power is the quotient between the energy and the compute time:

$$\overline{P} = P_{st} + (1/T_C) \times E_{dyn}, \tag{4.5}$$

$$\overline{P} = I_{BIAS} \times V_{dd} + (1/T_C) \times 2 \times C_{int} \times (\Delta V)^2, \tag{4.6}$$

$$\overline{P} = 500\text{nA} \times 1.2\text{v} + (1/1\mu\text{s}) \times 2 \times 250\text{fF} \times (0.5\text{v})^2, \tag{4.7}$$

$$\overline{P} = 13.1\mu\text{W}, \tag{4.8}$$

where T_C is the compute, time set to 1μ s. The MAC operation is considered to be a fused-multiply-and accumulate (FMAC) [78], where $a \leftarrow a + (b \times c)$ is carried out using full precision in the analog domain, and later re-quantized down to 8-bits by an external A/D converter (not considered in the energy calculation). Each CID row is capable of performing 32-fold 4-bit MACs in the stochastic domain. This is achieved by encoding the each 4-bit number along $2^4 = 16$ columns; since the array has 512 columns, then 32 operations can be performed in parallel per conversion. At this rate, 1MAC can be computed in $1\mu s/32 = 31.25$ns, which gives 32MMACs/s. Finally, the

Table 4.2 *Design specs for the pseudo-DRAM mixed-signal computational unit.*

Technology	55nm GlobalFoundries LPX
Core area	0.35mm^2
Core voltage	1.2v
Compute Speed	1MHz
Array dimensions	156×512
DRAM memory	10KB
Bit-Cell area	1.2 μm^2
Input precision	4-bits
Parameter precision	4-bits
Output precision	8-bits
Compute modality	Stochastic
Operation	Fused Multiply-and-Accumulate (MAC)
Operations per second	4.9 GMACs
Energy Efficiency	2.44 TMACs/W

efficiency is computed as:

$$E_{ff} = \frac{MACs/s}{\overline{P}}, \tag{4.9}$$

$$E_{ff} = 2.44\,[\text{TMACs/W}]. \tag{4.10}$$

4.4 Conclusion

A pseudo-DRAM Compute-in-Memory (CiM) test-chip was designed and fabricated in a CMOS 55nm technology. The has proven to demonstrate the usefulness of charge-computing by operating with voltage swings that are near the KTC thermal noise limits. The primitive computing element consists of a Charge-Injection-Device (CID), which to our knowledge has never been fabricated in a feature size smaller than 180nm. The DRAM crossbar allows for memory storage as well as analog computing in the stochastic domain. The fully custom architecture comprises a digital parallel input bus, and a digital tri-state output bus for memory recall, as well as digital peripheral glue circuits.

The wire-bonded chip was mounted on a custom Printed Circuit-Board (PCB), which in turn was connected to the back of an Opalkelly xem 7310 expansion board, which includes an Artix 7 FPGA. All on-chip circuits were controlled via the FPGA indirectly from custom Python API running on a host Linux PC. Since the test-chip does not include A/D converters, the Artix 7 FPGA on-chip SAR A/D was used. Test results gave insight into internal leakage phenomena and parasitic coupling of internal nodes. Gate-oxide tunneling was identified as the main source of leakage. A second source was identified as sub-threshold current in the access transistor, causing leakage in the opposite direction as the tunneling. This effect is seen in the time-domain output curves where the leakage slope decreases as a function of the input. Additionally, the effect of finite gain of the OTA also contributes to the change in slope, causing it to decrease as the input, X, increases.

Test results showed the CiM architecture was capable of accelerating multiplications using both Pulse-Width-Modulation and Stochastic coding techniques, discussed in this thesis. Post-processing shows the chip capable of computing 4.9GMACs/sec with an energy efficiency of 2.44TMACs/W.

Chapter 5

Suanpan: A Nano-Abacus Compute-in Memory (CiM) Processor Array

5.1 Introduction

The Nano-Abacus System-on-Chip (SoC) was developed as part of the DARPA-UPSIDE [60] program. As mentioned earlier, the goal of the program was to explore unconventional micro-architectures for improving real-time processing of video imagery. The system was designed to provide high computation bandwidth and energy efficiency, achieved by leveraging parallelism and unconventional computing paradigms. The full system comprises three Chip-Multi-Processors (CMPs), a high bandwidth 3D DiRAM [79], and a Xilinx Zynq 7 FPGA. The CMP dies, the FPGA, and the 3D DiRAM are to be integrated on silicon interposer, which acts as a passive host die interconnect.

The first part of this chapter describes system-level implementation details and exposes the CMP architecture. The former stresses the use of modular/reusable designs, which are interconnected via a Network-on-Chip (NoC) as well as the passive interconnect at the interposer level. In the second part of this chapter, one of the CMPs is described, which contains heterogeneous processing units, specifically, the

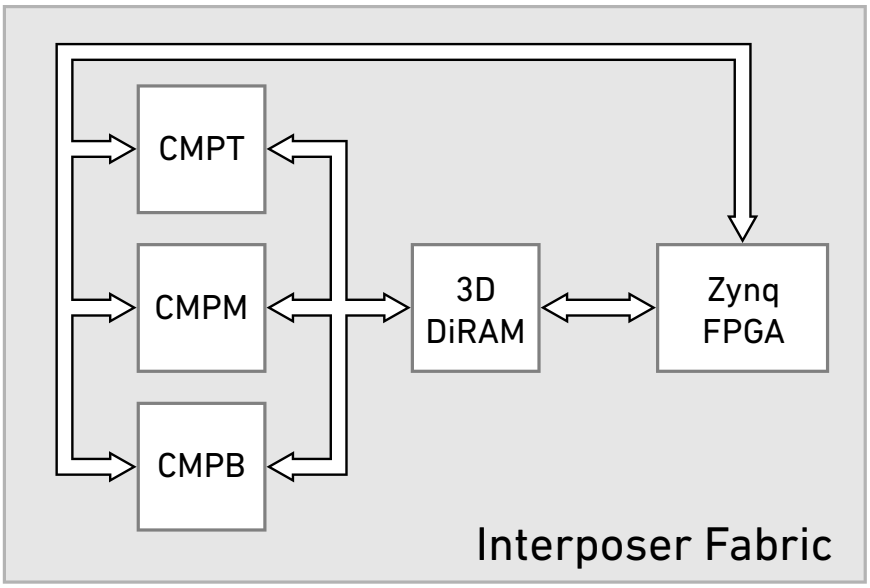


Figure 5.1 *Top view of the 2.5D Nano-Abacus Chip-Multi-Processor.*

Suanpan core. The word *Suanpan* is of Chinese origin and is used to describe an abacus used to perform basic arithmetic by counting discs that slide on small rods. The Suanpan CMP utilizes similar principles in order to perform in-memory arithmetic at the nanoscale. The concept of counting is leveraged by utilizing charge to represent quanta of data, which is added passively by reading the current that travels through an internal node. Non-Volatile Memory is of particular interest due to its demonstrated capability to perform arithmetic at low energy expense and will be the main focus for the remainder of the chapter.

5.1.1 Architecture of a 2.5D Nano-Abacus SoC

The 2.5D Nano-Abacus SoC comprises 3 Chip-Multi-Processors (CMPT, CMPM, CMPB), each consisting of the same I/O footprint occupying $12.13mm \times 17.16mm$ of die area in CMOS 55nm technology. The CMPs are mounted on a passive die interconnect, named *interposer*, and can communicate with each other, the FPGA, or the DiRAM. The size of the interposer is $50mm \times 64mm$ and was fabricated in a $1\mu m$ process. Figure 5.1 illustrates the three CMPs, along with the FPGA and DiRAM.

Each heterogeneous CMP consists of the following elements:

- An array of Processing-Units (PUs), which are specialized unconventional processors tailored to solve different tasks in typical image processing pipeline. Noteworthy, the CMP described in this section is heterogeneous, although only the NVM PU will be described.
- A two-dimensional Network-on-Chip (2D-NoC), in which Processing-Units can connect to nodes. The 2D-NoC is composed of:
  - A token ring (N1-NoC) [80], used to shuttle high-speed data to/from the PUs.
  - A 2D mesh network (N2-NoC), used to shuttle data between PUs.
- A General Purpose I/O (GPIO) interface, which is used to configure PUs and other on-chip parameters via the FPGA.
- A high bandwidth interface, which allows the N1-NoC to communicate with DiRAM and send large amounts of data to/from main memory.

Figure 5.2 shows a conceptual diagram of the CMP, showing its main interfaces and processing units that are connected to different nodes on the NoC. The N2 topology has M rows and N columns, where the intersections are composed of a network router, such as shown in Fig. 5.3. The router allows each processing unit to connect to both N1 and N2 NoC levels via an Interface-Control-Unit (ICU). The former is a dedicated processor with eight instructions; the ICU allows multiple processing units to work in tandem, enabling complex tasks involving large amounts of data with the need to access main memory or the FPGA. The ICU is programmed via N2 through a configuration unit, which can be either the M0 processors or the FPGA.

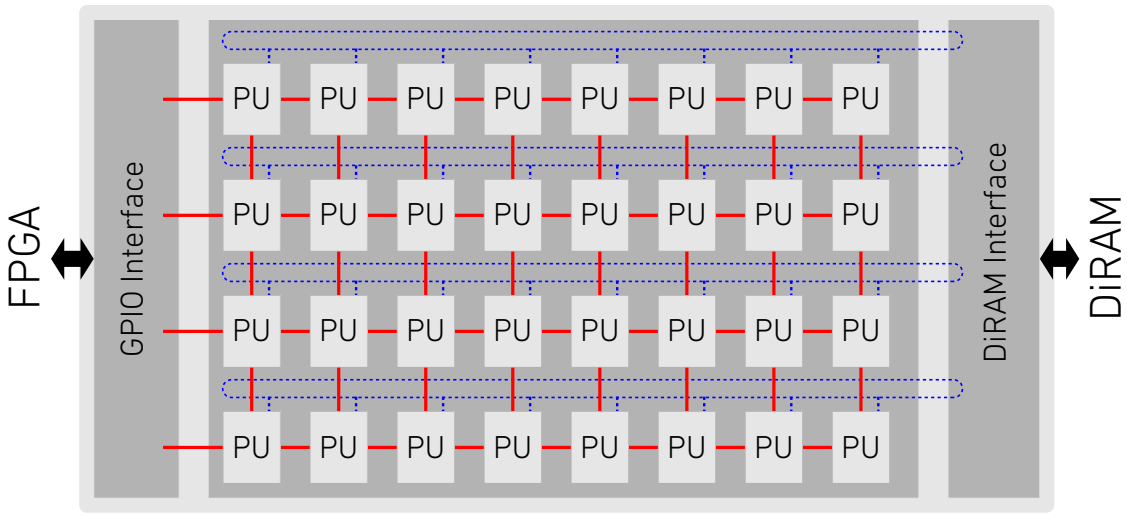


Figure 5.2 *Network-on-Chip, where different processing units are connected to nodes. The token ring (N1) is shown in blue, and the mesh network is drawn in red (N2).*

5.1.2 Memory Scheme

On the PU side, data transfers obey the AHB-Lite protocol, which is described in detail in [81]. High throughput is achieved by using wide 256-bit, input/output buses. Two additionally signals are used to initiate or finalize an internal PU process, e.g., that a computation has reached completion. Conversely, the NoC side has two ports to communicate with N1 and N2. Finally, data transfers between main memory and the processing units take place via a 256-bit width bus, which is synchronized via four-phase handshake, thus the sit on different clock domains.

5.1.3 Chip-Multi-Processor: *Suanpan Tablet*

Within the scope of this work, the focus was to co-design the heterogeneous chip multiprocessor (CMP), which is called *Suanpan Tablet*. Figure 5.4 shows a top level layout, where around 30% of the chip (on the left) is not shown because it was occupied by University of California, Santa Barbara. The figure shows the different blocks of the CMP, listed below:

- An ARM M0 processor that is used for house-keeping related tasks. The M0 is

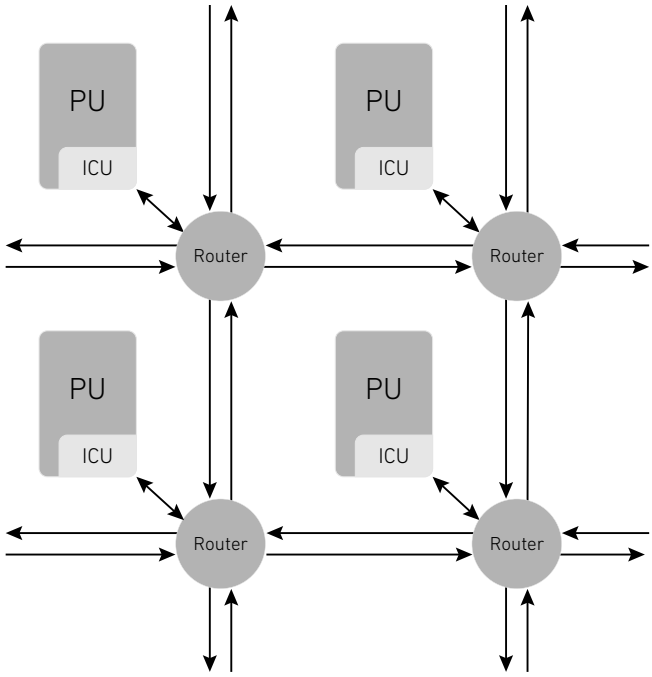


Figure 5.3 *Network-on-Chip node, which includes a network router that provides connectivity between the PU and N1, N2 NoC levels.*

set to boot via UART interface and is useful to perform pre/post processing of data that travels through the NoC.

- An auxiliary unit, consisting of oscillators for on-chip clock generation, and band gap references for on-chip bias generation.
- PWL processors, used to perform morphological operations on images. This block occupies two node locations on the NoC.
- Multiply-and-Accumulate (MAC) units, which are digital, fixed point arithmetic computational units optimized for low power operation.
- On-chip memory, which is denominated *CACHE*, although it only behaves as a passive random-access-memory.
- A DiRAM interface, which is a custom designed to communicate with the external DiRAM.

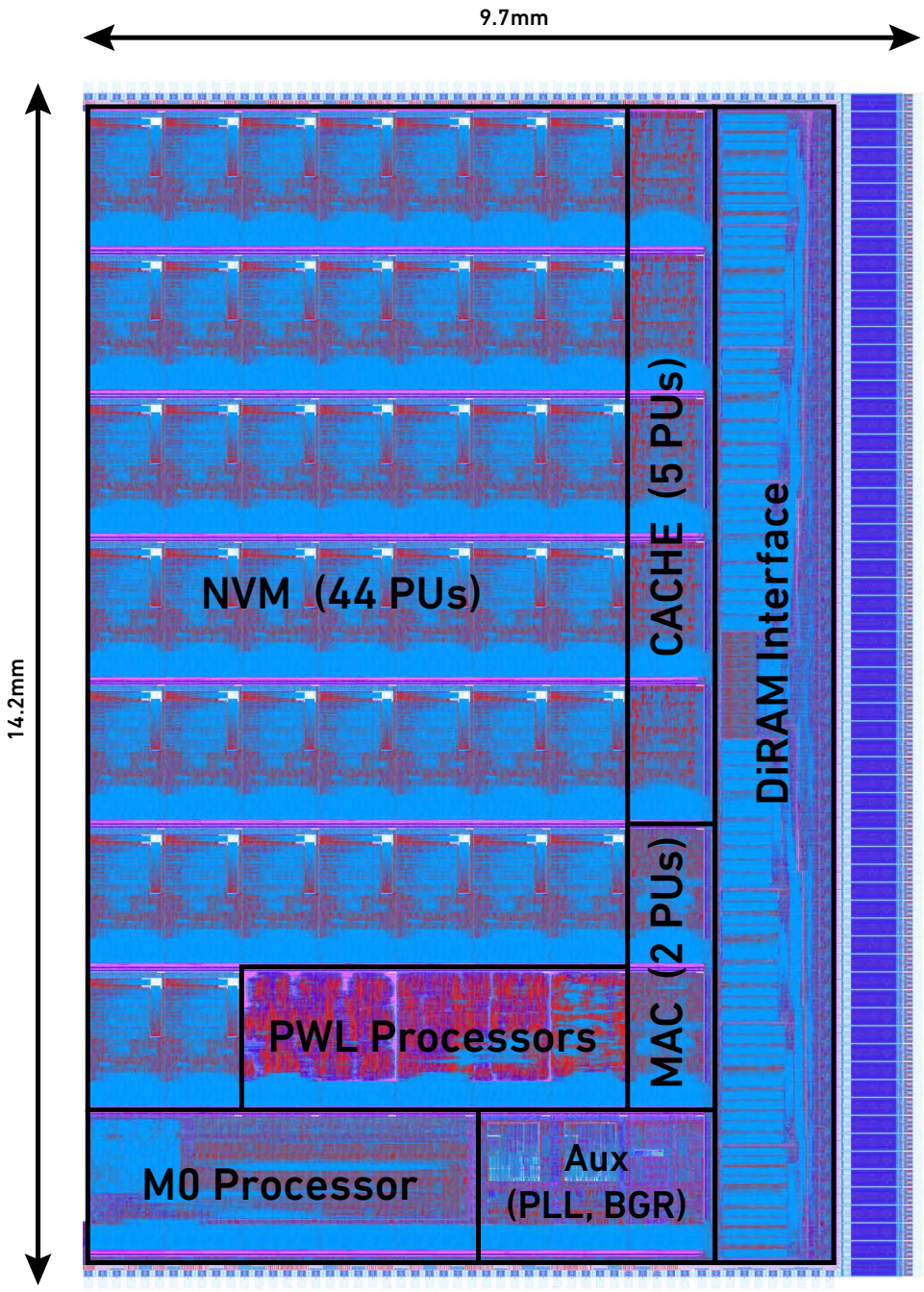


Figure 5.4 *Top level layout of the heterogeneous CMP Network-on-Chip node, which includes a network router that provides connectivity between the PU and N1, N2 NoC levels.*

- Non-Volatile Memory (NVM) processing units, which are computational memories used to perform on-chip inference using low-precision fixed point arithmetic. Additionally, the NVM units can be used as memory to store local parameters.

The remainder of this chapter will focus specifically on the design of the NVM processing unit.

5.2 A Mixed-Signal Compute-in-Memory Processor using NVM Primitives

A mixed-signal processor using SST non-volatile memory (ESF3 NVM) was designed and implemented as a part of the Suanpan, heterogeneous processor array. On Suanpan, there are 44 identical NVM processing units (PUs) that can communicate with each other, other processors on the CMP, and main memory through the NoC. A top level block diagram of the NVM processing unit is shown in Fig. 5.5. Blocks enclosed in light-blue correspond to digital synthesized circuits, while the rest comprises full-custom manual mixed signal circuits. Notice that the computational core is composed of a 128×512 NVM crossbar (top right). Inputs to the crossbar are connected horizontally from left to right, while outputs are wired vertically from top to bottom. The NVM CiM core is capable of processing 512 vector-vector multiplications (VVM) in parallel, where parameters are stored locally in multilevel floating gate cells. The PU

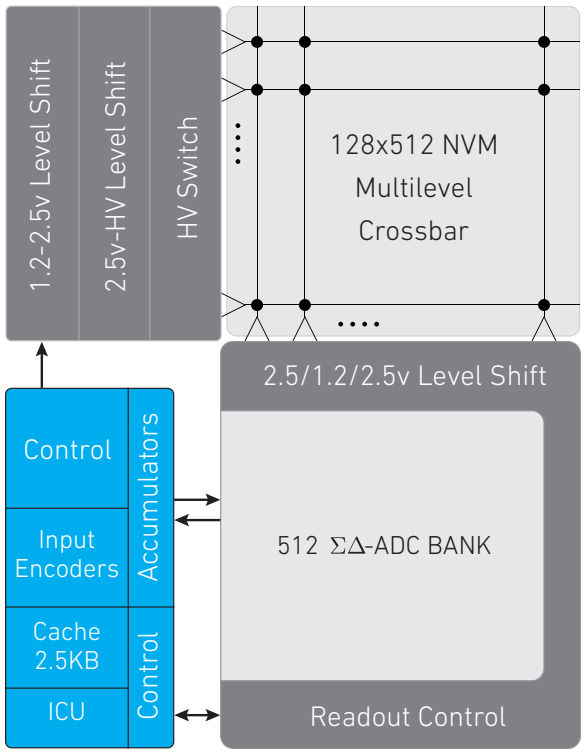


Figure 5.5 *Top level block diagram of the NVM processing unit.*

disposes an Interface-Control-Unit (ICU), used to communicate with N1 and N2 NoC, and local distributed CACHE RAM memory. The former is used to store control parameters, such as A/D output precision, and non-uniformity correction (NUC) parameters for the computation. Additionally, the digital-synthesized block contains all auxiliary control and compute circuits, such as encoders, accumulators, bit-masks, and decoder-multiplexers for all AHB-Lite memory mapped registers.

5.2.1 Mixed-Signal Core Architecture

The core of the analog block is a 128x512 NVM crossbar capable of computing up to 512 unsigned, or 256 signed inner products simultaneously. Figure 5.6 illustrates

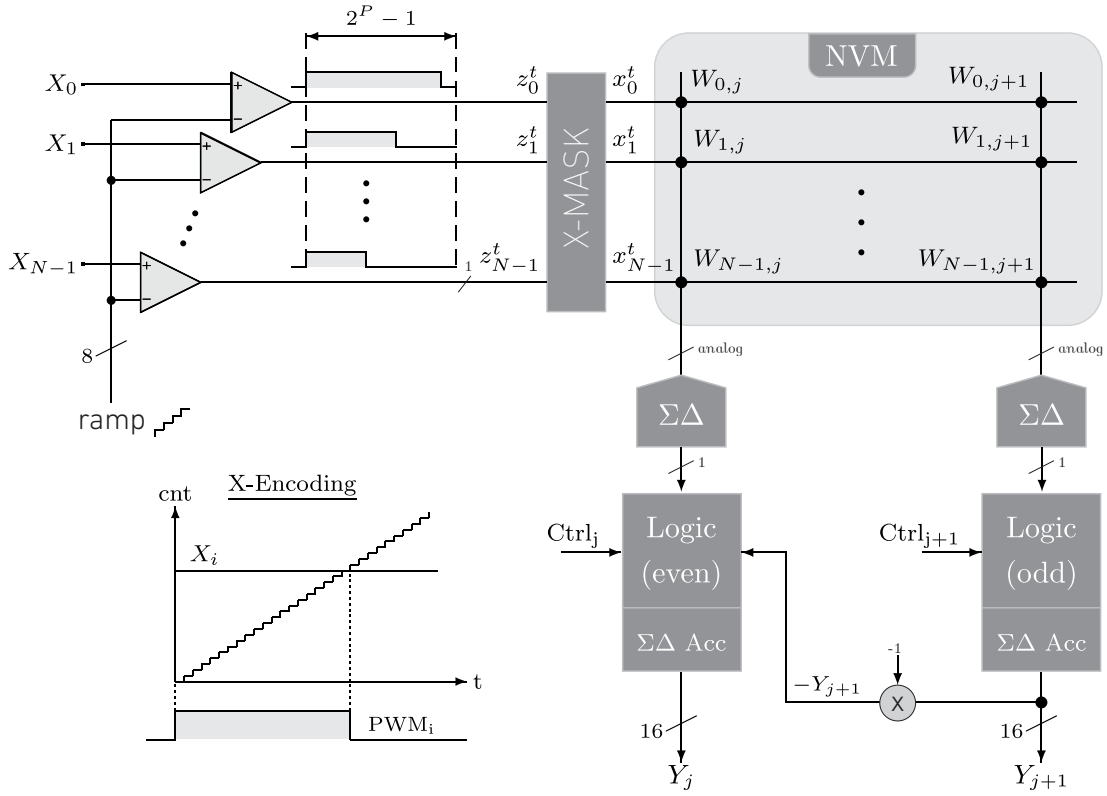


Figure 5.6 *NVM mixed-signal core architecture. One and two-quadrant Vector-Matrix Multiplication (VMM) using multi-bit NVM. The signed (two-quadrant) operation utilizes two columns, while the unsigned operation only uses only one column. The bottom left of the figure shows the output of a single encoder, which generates a PWM signal that is partitioned into* $2^P - 1$ *compute-cycles. During each compute compute-cycle, the output accumulators (*$\Sigma\Delta$*) run for* $M = 2^{L+1}$ *clock cycles in order to achieve an* $L-$*bit result. The X-Mask block performs a mask operation on the input.*

a typical functional block diagram of a column-wise computation. The mixed-signal core is a digital-input, analog-processing, digital-output system. This architecture is based on the digital NVM computation model described in Section 3.3.2.

The D/A conversion is accomplished by using a time domain pulse width modulation (PWM), which is achieved by comparing digital inputs against an 8-bit ramp. The ramp is capable of counting in programmable increments, thus yielding flexible precision D/A conversions. The analog result from the computation is represented as a nodal current, which is later converted to digital with a first-order Sigma-Delta ($\Sigma\Delta$) modulator and an accumulator. Since the D/A conversion is carried out in time (as opposed to space), for each input, the $\Sigma\Delta$ must convert the output computation and store the result in an accumulator. Once a final result has been computed, a bus master can fetch the result, the controller can reset the accumulator, and a new computation can start. The system is capable of carrying out 1-quadrant and 2-quadrant VMMs, explained below.

One-quadrant Unsigned Vector-Matrix Multiplication (VMM)

Following Fig. 5.6, the unsigned operation is described below:

- First, a digital vector $\mathbf{X}$ is compared to a ramp, thus encoding its value into a N-dimension pulse-width-modulation (PWM) signal, $\mathbf{z^t}$, of duration 2^{P-1} compute cycles. The number of bits for encoding $\mathbf{X}$ is denoted by P, and the superscript t refers to time.

- Next, at each compute-step, t_k, a mask operation is performed on $\mathbf{z^t}$, yielding a $1 \times N - bit$ vector with elements, $x_i^{t_k} = z_i^{t_k} \cdot m_i$, where m_i is the masking bit of the i^{th} row. Subsequently, $x_i^{t_k}$ is wired to the NVM crossbar controlling the on-off state of the i^{th} row.

- Each solid dot in the crossbar symbolizes a floating gate programmed with a

weight $W_{i,j}$, which represents the digital value of the output current of the cell when it's active. NVM cells are active-high controlled by $x_i^{t_k}$, following the expression:

$$I_{i,j}^{t_k} = \begin{cases} W_{ij} & x_i^{t_k} = 1 \\ 0 & x_i^{t_k} = 0. \end{cases} \tag{5.1}$$

- All output currents are intrinsically added in the column, yielding:

$$I_j^{t_k} = \sum_{i=0}^{N-1} I_{i,j}^{t_k} \tag{5.2}$$

$$I_j^{t_k} = \sum_{i=0}^{N-1} W_{i,j}\, x_i^{t_k}. \tag{5.3}$$

- Next, the total output current per column is converted to a pulse density stream by means of a first order sigma-delta ($\Sigma\Delta$) modulator.

- Finally, this unary stream is accumulated with variable gain, $\mu_j^{t_k}$, and initial offset, κ_j, over an arbitrary number of clock cycles, M_c, which determines the output precision. The expression at the end of a compute cycle, t_k, is shown below:

$$Y_j^{t_k} = \kappa_j + y_j^{t_{k-1}} + y_j^{t_k}, \text{ where} \tag{5.4}$$

$$y_j^{t_k} = \sum_{i=0}^{N-1} \mu_j^{t_k} W_{i,j} x_i^{t_k}. \tag{5.5}$$

- If weights $W_{i,j}$ represent a K-bit value, then $y_j^{t_k}$ yields a $1 \times K - bit$ inner product whose result is stored in the sigma-delta accumulator. Since the accumulator is not reset between compute cycles, then repeating this computation over $2^P - 1$ cycles yields a full $P \times K - bit$ inner product with variable gain and an initial

offset, given by:

$$\mathbf{Y_j} = \kappa_j + \sum_{k=0}^{2^P-1} y_j^t \tag{5.6}$$

$$\mathbf{Y_j} = \kappa_j + \sum_{k=0}^{2^P-1} \sum_{i=0}^{N-1} \mu_j^{t_k} W_{i,j} x_i^{t_k} \tag{5.7}$$

$$\mathbf{Y_j} = \kappa_j + \sum_{k=0}^{2^P-1} \sum_{i=0}^{N-1} \mu_j^{t_k} W_{i,j} (z_i^{t_k} \cdot m_i). \tag{5.8}$$

- At this point, each column has performed the unsigned operation described in Eq. 5.8, conveying a full 128×512 vector-matrix multiplication between a masked input vector $\mathbf{X}$ and a weight matrix $\mathbf{W}$. Additionally, an initial bias and variable gain per compute-cycle is utilized.

Noteworthy, since each bit-cell is capable of outputting $\sim 30\mu A$, it is imperative that the mask, m, is utilized correctly, by enabling only a few rows at a time. If all rows are active, each array can easily draw $2A$, which would likely cause permanent damage to the system. For this same reason, power-up conditions have been carefully considered.

Two-quadrant Signed Vector-Matrix Multiplication (VMM)

Following the procedure described above, a two-quadrant signed 128×256 VMM operation may be performed by subtracting the outputs of adjacent columns, such as depicted in Fig. 5.6. One possible way to achieve this result is by storing positive values of W in even columns, and the magnitude of the negative weights in odd columns. At the end of the last compute cycle, column outputs Y_j and Y_{j+1} are subtracted, as

shown below:

$$\mathbf{Y_j} = \kappa_j + \sum_{k=0}^{2^P-1} \sum_{W_{i,j} \geq 0} \mu_j^{t_k} W_{i,j} (z_i^{t_k} \cdot m_i) \tag{5.9}$$

$$\mathbf{Y_{j+1}} = \kappa_j + \sum_{k=0}^{2^P-1} \sum_{W_{i,j} < 0} \mu_j^{t_k} |W_{i,j}| (z_i^{t_k} \cdot m_i) \tag{5.10}$$

$$\mathbf{Q_n} = \mathbf{Y_j} - \mathbf{Y_{j+1}} \tag{5.11}$$

$$\mathbf{Q_n} = \kappa_j + \sum_{k=0}^{2^P-1} \sum_{i=0}^{N-1} \mu_j^{t_k} W_{i,j} (z_i^{t_k} \cdot m_i). \tag{5.12}$$

Interestingly, the offset and gain parameters may be used as a non-uniformity correction computation block, where the multiplication is performed with very low power, thus it is merely the result of applying gain to the accumulator, which in reality is embodied as the increment. Additionally, this type of structure may be viewed as neuromorphic, where rows are weighed dentritic inputs, and the columns are axons, where the output may be scaled and biased.

5.2.2 Analog and Mixed-Signal Circuits

In this section, low level analog and mixed signal circuits pertaining the the NVM core are described.

NVM bit-cell

The NVM bit-cell is an SST proprietary third generation floating gate. For cross-section and physical information please refer to Section 3.2. The bit-cell is a six terminal device, with three gate terminals: word-line (WL), coupling-gate (CG) and erase-gate (EG), two diffusion terminals: bit-line (BL) and source-line (SL), and a bulk terminal connected to the common p-substrate. During read operation, current flows into the floating gate and is measured with a BL voltage of approximately 0.8[v]. Higher BL voltage will produce a larger current, while lower WL voltage will produce a lower current. Erase operation relies on high voltage on the EG in order to tunnel electrons out of the floating gate. Finally, programming is achieved by hot carrier

Table 5.1 *Truth tables for row switches. "S0" and "S1" are prefixes to the names in Fig. 5.7, e.g. S0_eg, S1_eg.*

S1	S0	CG	SL	EG	WL
0	0	2.5	0.5	4.5	VWLP
0	1	GND	GND	GND	GND
1	0	HV	4.5	HV	2.5
1	1	GND	GND	0.5	GND

injection: hot electrons are accelerated from SL to BL (currents of approximately $1\mu A$) and attracted to the floating-gate by means of a large electric field produced by the CG. Hot electrons may be produced either by a current source or by potential difference between SL-BL. Cell operations are summarized in Table 3.2 and Table 3.1.

Interestingly, the direction of the current is reversed between *read* and *program* modes, therefore traditional programming where the device is connected in the in the feedback loop of an amplifier does not apply. Conversely, gradual or multilevel programming is achieved by iteratively pulsing small amounts of currents from SL to BL followed by a *read* operation. Normally, this is accomplished by pulsing the CG, yet it may be possible to pulse to BL at the expense of undesired disturbance to adjacent cells.

Row Switches: WL, EG, CG, SL

In the previous section, NVM switching activity was mentioned, however, no implementation details were revealed. It is important to note that several signals require voltage levels greater than 2.5[v]. The GF55LPX process provides high voltage devices with thick gate oxide ($\sim 20nm$) and lightly doped drains, allowing high voltage operation under controlled conditions. Figure 5.7 illustrates all control switches for horizontal, i.e. row signals: WL, EG, CG, SL. Every pair of rows of an NVM array composes a page, which allows the same control circuitry to share several signals in order to save space. Interface between digital (0-1.2v) control signals and high voltage gates

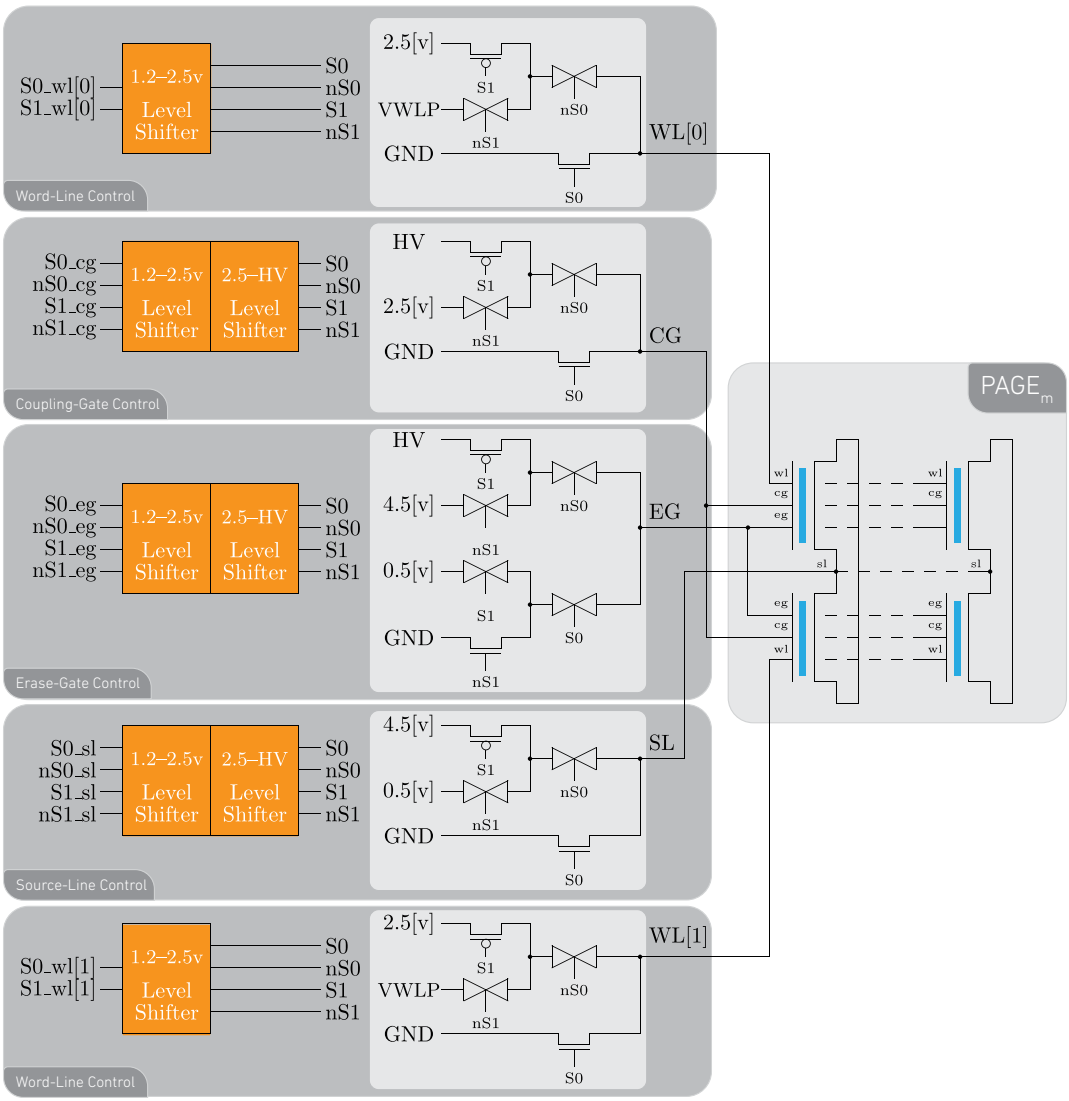


Figure 5.7 *Circuitry to drive one pair NVM of rows, where the on-state is mutually exclusive. Signals with the same name within a pair of dashed-lines are connected to each-other.*

is achieved by two stages of level shifting. The first stage, shifts 1.2v signals to a 2.5v domain using a standard 4T circuit and the second stage, shifts the respective 2.5v signals to the HV domain. From the figure, the left-most control signals (e.g. S0_cg and S1_cg) are provided by the digital synthesized portion of the chip and are converted to their respective power domain as explained earlier. Signals that do not demand HV voltage only require one-stage of level shifting, specifically the word-line. Level-shifters control the HV switches, which in turn control all row signals in the

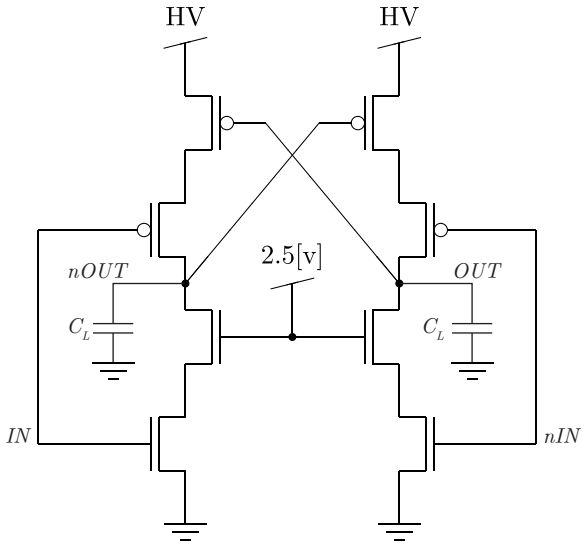


Figure 5.8 *An eight-transistor level-shifter which allows interface between 2.5[v] and HV signals, where HV>2.5[v]. The cascoding effect mitigates the effect of high V_{DS} drops during transient operation.*

NVM array. The circuitry depicted in the figure is sufficient for controlling one pair of rows, thus the full processing unit requires 64 of these blocks. A truth table for digital inputs, $S0$, $S1$, is shown in the left of Table 5.1.

In this work, HV (High Voltage) is referred to any power supply or bias above 2.5v. Nodes in the NVM array may be either highly capacitive (gates) or low impedance (sources and drains), therefore high transient currents are expected when those nodes change state. Hence, careful circuit design and sizing was taken into consideration accounting for two limiting factors: (i) static current must be less than $20\mu A/\mu m$, and (ii) transient current must be less than $200\mu A/\mu m$. A circuit for the second stage of level-shifting is shown in Fig. 5.8, where the effective load, C_L, and relative load balance between its outputs dictate the total transient current in its branches. In order to decouple the level shifters from the NVM array, a conservative design approach was taken; level shifters where used to drive switches (pull-ups, pull-downs, and full transmission gates), which in turn connect NVM internal nodes to low impedance power supplies. Transistors in the level shifters were designed to be minimum size, while switches were designed carefully to comply with the current density limits.

A final note on control circuits for the row design. In principle, only one high voltage control block is needed for the array, and the switches could be used to multiplex the control over all pages. However, it was desired to have full-parallel control of the array due to potential unknown limitations of the multiplexed approach. Additionally, during the compute phase, it is necessary to have parallel control over word-lines, thus several NVM rows mush active simultaneously. This circuitry does not require voltages higher than 2.5v, therefore the area impact of replication is not that severe as the that for the other control signals.

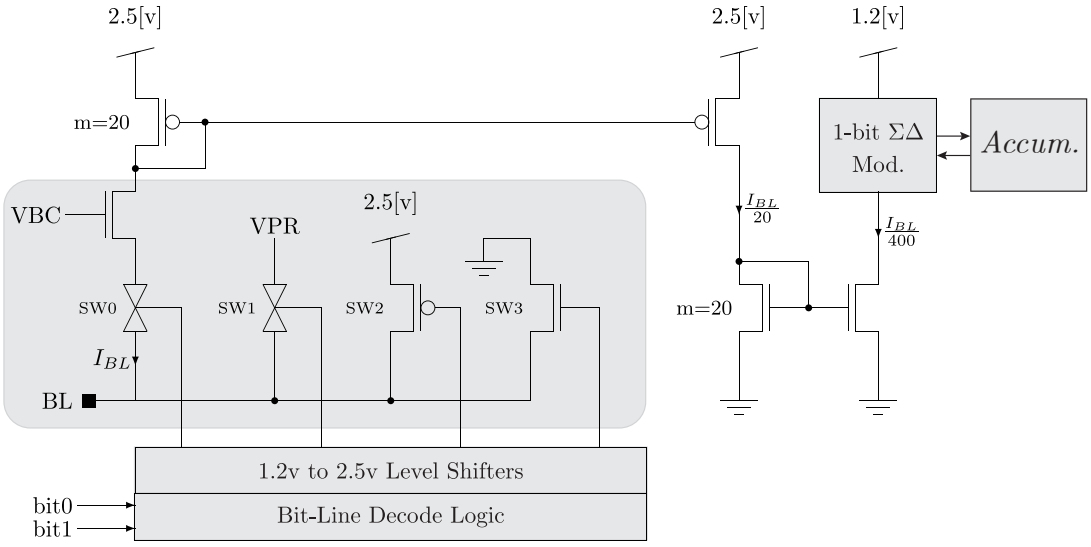


Figure 5.9 *Bit-line control and readout circuits. Note that the bit-line readout current is mirrored twice, scaling it by a factor of 1/400.*

Table 5.2 *Truth table for NVM bit-line control. This table references signals described in Fig. 5.9, where each column utilizes two control bits.*

bit1	bit0	Mode	Active Switch
0	0	Erase	SW3
0	1	Read	SW0
1	0	Program Select	SW1
1	1	Program Unselect	SW2

Column Switches: BL

The NVM array shares bit-lines on a column-by-column basis, therefore one control/readout circuit per BL is required. Figure 5.9 shows all related switches that interact with a bit-line, which are active-exclusive, and controlled by two signals: $bit0$ and $bit1$. In order to decongest routing lanes in the layout, signal decoding is performed locally according to Table 5.2. From Fig. 5.9, bias VBC serves two different purposes: (i) throttling bit-line current during read operation, and (ii) $VBC = 0$ during power-up cycle forcing the output current to zero. The latter is especially important since at power-up, the state of the control registers is unknown, hence a high surge current could damage the chip.

During read operation, the bit-line current is mirrored twice: the first mirror is p-type and belongs to the 2.5v power domain, and the second mirror is n-type and routes the current into the input branch of the A/D converter, which belongs to the 1.2v power domain. Noteworthy, shifting power domains for the A/D converter results in a lower available voltage swing, however, it reduces its size by a factor of at least two, and enables direct interaction of the sigma-delta modulator with the synthesized HDL block.

First Order Sigma-Delta Converter

During a *read* operation, the bit-line current is divided by 400 (by mirroring twice) and then steered to the input branch of a first order current-mode sigma-delta ($\Sigma\Delta$) modulator, which is shown in Fig. 5.10. The diagram shows the different stages in the circuit; portions enclosed in light gray correspond to manual layout and devices in light blue pertain to semi-custom digital circuits that were integrated in to the place and route (P&R) flow. Conceptually, the input current is used to modulate the pulse density of a binary stream, which is later wired to the control input of a digital accumulator. The first order sigma-delta was chosen due to several factors,

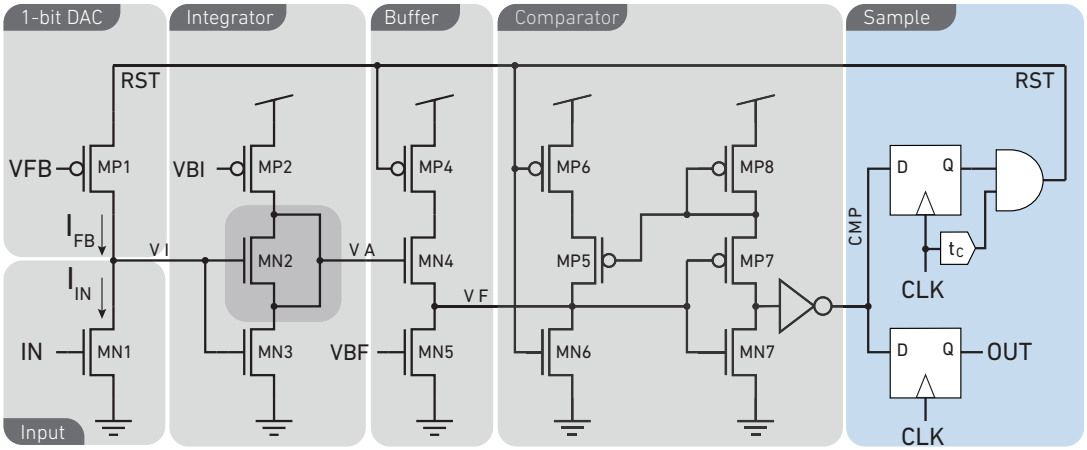


Figure 5.10 *Schematic circuit diagram of sigma-delta modulator for A/D conversion. The output is a binary sequence whose pulse density represents the analog value of the current at the input. The block in light blue shows the portion of the circuit that interacts with the digital domain.*

listed below:

- **Signal-to-Noise Ratio (SNR):** oversampling a sigma-delta results in high-pass filtering of the white, Gaussian, uncorrelated quantization noise. This effect is called noise-shaping [82], and is easily shown due to the presence of an integrator in the direct-loop of the sigma-delta. The former is seen as derivative in the noise transfer function (NTF), and effectively improves the SNR in base-band signals, such as the nodal-current being converted to digital in this work. Additionally, the integration capacitor is not reset between conversions, therefore the charge residue is used for the next conversion, and effectively creates a long-term current-mode time-averaging effect. As shown in Section 5.2.2, the averaging effect helps reduce the shot noise, allowing subthreshold currents to represent 6-bits (or more) of binary data.

- **Size:** a charge-current mode sigma-delta is used, which results in a small number of transistors, which in turn reduces area. Additionally, charge mode feedback along with high sampling frequencies allow the integration capacitor to small.

- **Power:** The current-mode sigma-delta requires only a few biases, which operate

in the same order of magnitude as the input current. Since the input is divided by a factor of 400, only a few hundred nano-amps are needed to operate the entire circuit. Additionally, the dynamic power, CV^2, is minimized due to the small size of C. A power breakdown of the circuit is shown later in this section.

- **Simplicity:** a unary encoding of the current is simple and allows for digital synthesized post-processing to take place. Accumulation and decimation are carried out on-demand, yielding different conversion precisions as seen fit.

- **Linearity:** first order sigma-deltas with a 1-bit DAC in the feedback loop ensures linearity and monotonicity.

The circuit in Fig. 5.10 consists of three main stages: (i) Integration, (ii) comparison, and (iii) sample and feedback.

i) **Integration:** upon a falling edge of RST, the input current is integrated in the form of charge via a trans-impedance amplifier (MN3) onto the integrating capacitor (MN2), yielding an integration signal VA. Next, VA is buffered via MN4 to the input of a clocked comparator.

ii) **Comparison:** the comparator consists of an inverter (MN7, MP7) with positive current feedback (MP8, MP5), and was inspired by work in [83]. During reset, MN6 pulls the comparison node (VF) to ground and MP6 opens the positive feedback branch. Next, VF increases proportionally to VA until the threshold is reached. When the comparator triggers, the signal CMP goes high and is sampled on the next rising edge of the system clock. The comparator threshold depends on sizing and was set to be approximately vdd/2. It is important to note that it may take several clock cycles before the comparator triggers.

iii) **Sample and feedback:** the output of the comparator, CMP, is sampled (bottom register) using the system clock, of period T. The output of the register,

OUT, is used to control the increment of a digital accumulator (not shown in the figure). Additionally, when $CMP = 1$, a reset pulse (RST) of duration T/2 is generated (top register). From the figure, the delay block (ΔT_C) is set to be larger than the propagation delay of the register, thus RST is guaranteed to be glitch-free. The feedback consists of a 1-bit D/A converter, where the feedback bias, VFB, is set to the maximum input range. In order to ensure the same quantum of charge es injected in the circuit, the feedback loop is chopped with the clock. In this way, any non-linearity in the feedback mechanism is replicated equally for different regions of operation.

The key to correct functionality relies on three main factors: (i) feedback bias, (ii) the clock period, and (iii) the size of the capacitor.

i) **Feedback bias (VFB):** under the conditions of maximum input current, I_{IN}^{max}, the charge removed from the integration capacitor will be:

$$Q_{IN}^{max} = I_{IN}^{max}\, T \tag{5.13}$$

where T is the clock period. Since the desired output is a sequence of consecutive ones, the comparator should trigger every clock cycle, yielding a chopped reset signal of pulse width $T/2$. Under these conditions, we equate the total feedback charge to be equal to the input charge and solve for the feedback current (I_{FB}):

$$Q_{FB} = I_{FB}\,(T/2) \tag{5.14}$$

$$Q_{FB} = Q_{IN}^{max} \tag{5.15}$$

$$I_{FB}\,(T/2) = I_{IN}^{max}\, T \tag{5.16}$$

$$I_{FB} = 2\, I_{IN}^{max}. \tag{5.17}$$

From the equation above, the feedback bias VFB should be such that $I_{FB} = 2\, I_{IN}^{max}$. The remaining biases should be in the order of magnitude of I_{FB}, so

that the slew-rate of the internal capacitive nodes can keep up with the input. One of the advantages of this circuit is that the input current range may be changed by scaling the feedback current, and this does not affect the conversion speed (which is the case of other current-mode converters).

ii) **Clock period (T):** the sigma-delta circuit is designed such that the voltage fluctuations on the node VA are small enough that the amplifier doesn't reach the power rails. From Eq. 5.18 it can be seen that a larger clock period, T, would yield a larger voltage swing. For this chip, the lowest clock speed was set to be 20MHz.

$$v_C(t_0 + T) = v_C(t_0) + I_{IN}\,(T/C). \tag{5.18}$$

iii) **Capacitor size (C):** it can be seen from Eq. 5.18 that there exists a trade-off between clock speed and capacitor size. A larger capacitor allows the circuit to operate at lower frequencies, which may be desirable. However, there is an area trade-off; the integration capacitor in this chip was achieved by connecting 30 N-MOSCAPS in parallel, amounting to approximately $50fF$ in total.

Despite the advantages of the sigma-delta circuit, it is important to notice that unary data-converters take at minimum 2^{BITS} cycles to resolve an answer. Hence, high precision conversions will take exponentially higher conversion times, which must be considered at the system and algorithm levels. For this reason, most conversions in this work have been accomplished at 4-bit precision.

Efficiency of the sigma-delta converter

The sigma-delta was designed to operate at a maximum frequency of 300MHz for an input current of $\sim 20nA$. In order to analyze the efficiency, the energy is broken down into its basic components: static and dynamic. The static current on all biased

branches amounts to approximately $100nA$, which yields $120nW$ @ $v_{dd} = 1.2v$. The dynamic power is broken down into three main components:

- $\Sigma\Delta$ capacitor: assuming that the total voltage swing on the capacitor is $\Delta V_C = v_{dd}/10 = 120mV$, the energy to charge the capacitor is: $E_C = C(\Delta V_C)^2 = 50fF \times (0.12v^2) = 0.72fJ$.

- $\Sigma\Delta$ comparator: assuming a total node capacitance of $C_{comp} = 5fF$, $E_{comp} = 7.2fJ$.

- Sampling circuit: assuming a total node capacitance of $C_{samp} = 10fF$, $E_{samp} = 14.4fJ$.

Assuming an input of 50% of the maximum range, the comparator will switch half the time, and the input capacitance of the registers (and the delay block) will be switching every clock cycle. Table 5.3 shows an energy breakdown of the circuit, and Fig. 5.11 shows the energy distribution for the different components at an 8-bit conversion. From the pie chart, it can be seen that the dynamic power of the circuit is dominant due to the presence of high switching activity in the sampling-circuit. Furthermore, the static power that accounts for the biasing is several orders of magnitude lower than the dynamic power. A further improvement to this topology would involve careful design of the feedback mechanism, perhaps by incorporating an asynchronous self-reset and accumulator.

Noise Analysis

Current-mode A/D conversion is subject to shot noise, which comes from the discrete nature of charge when it crosses a potential barrier [76]. The standard-deviation, σ_I, of shot noise follows a Poisson process, yielding the following expression:

Table 5.3 *Energy distribution for first-order sigma delta modulator. Clock speed is 300MHz, duty cycle is 50%.*

Bits	E samp [J]	E comp [J]	E cap [J]	E static [J]	E total [J]	E total [fJ]
8	3.6864E-12	9.216E-13	9.216E-14	4E-16	4.70056E-12	4700.56
7	1.8432E-12	4.608E-13	4.608E-14	4E-16	2.35048E-12	2350.48
6	9.216E-13	2.304E-13	2.304E-14	4E-16	1.17544E-12	1175.44
5	4.608E-13	1.152E-13	1.152E-14	4E-16	5.8792E-13	587.92
4	2.304E-13	5.76E-14	5.76E-15	4E-16	2.9416E-13	294.16
3	1.152E-13	2.88E-14	2.88E-15	4E-16	1.4728E-13	147.28
2	5.76E-14	1.44E-14	1.44E-15	4E-16	7.384E-14	73.84

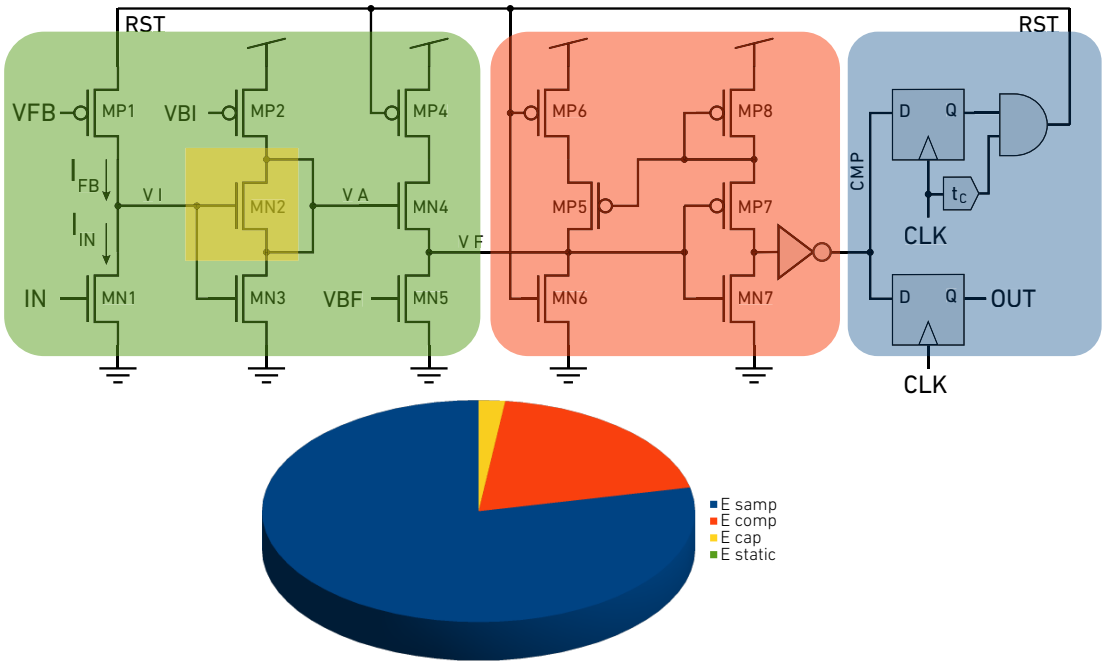


Figure 5.11 *Energy distribution for the first-order sigma-delta modulator.*

$$\sigma_I = \sqrt{2qI\Delta f}, \tag{5.19}$$

where q is the charge of the electron (considered positive), I is the mean current, and Δf is the single sided bandwidth of the signal. In our application, an $N-bit$ result is achieved by averaging current over 2^N clock cycles in the sigma-delta converter. Henceforth, the bandwidth is $\Delta f = f_{clk}/(2^N)$. This result is also intuitive, thus the more we average in the time domain, a smaller bandwidth is considered in the

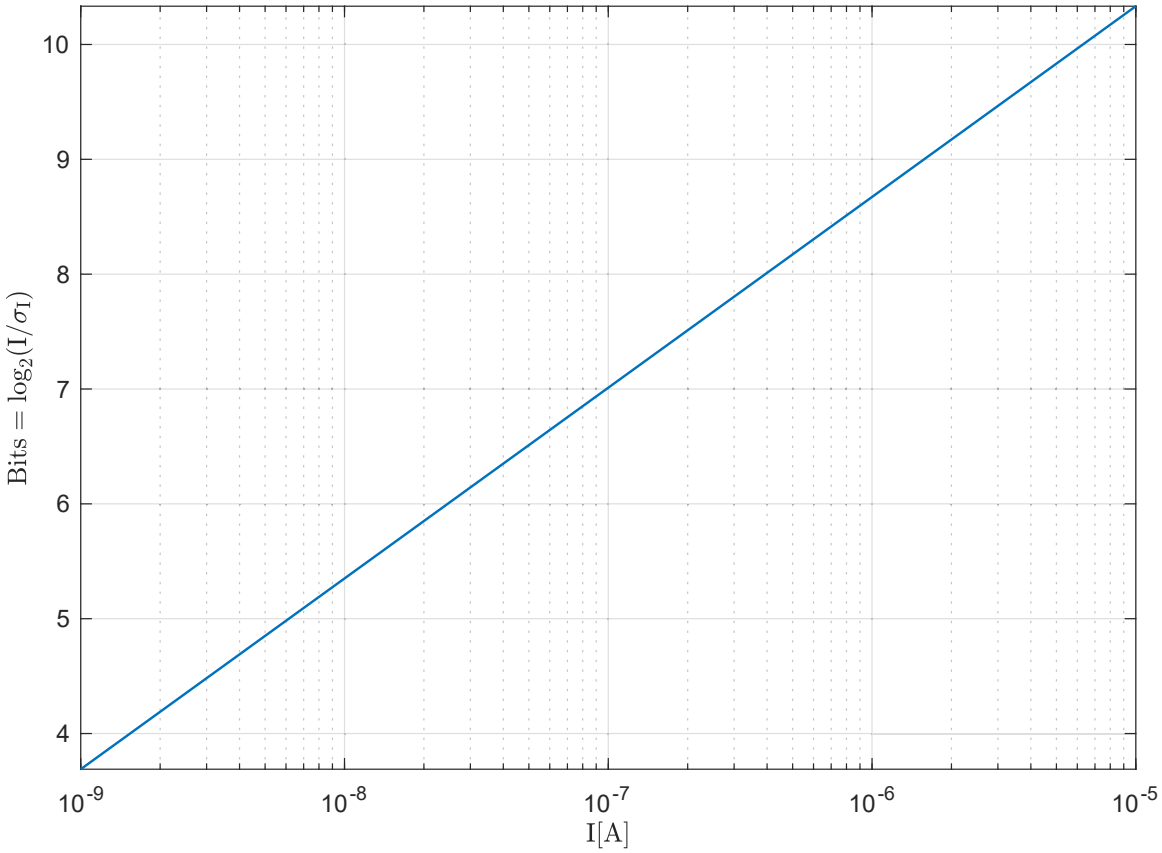


Figure 5.12 *Bits represented in when subject to shot noise. Notice that the x-axis is semilogarithmic, spanning from 1nA to 10μA. The graph was obtained by assuming a clock frequency of 300MHz, and a conversion precision of 4-bits, which implies 16 clock cycles per conversion.*

frequency domain. Moreover, the SNR is given by:

$$SNR = \frac{I}{\sigma_I}, \tag{5.20}$$

$$SNR = \frac{I}{\sqrt{I}} \frac{1}{\sqrt{2q\Delta f}}, \tag{5.21}$$

$$SNR = \frac{I}{\sqrt{I}} \frac{2^{N/2}}{\sqrt{2q\, f_{clk}}}. \tag{5.22}$$

From an information perspective, the number of binary bits that can be represented in a noisy current will be $BITS = \log_2(SNR)$. Figure 5.12 shows a semilog-x plot of the number of bits as a function of the mean current. The integration time was 16 clocks @ 300MHz, yielding a 4-bit result. Therefore, for a nominal current of 25nA, the sigma-delta is capable of resolving around 6-bits. This is an important result, thus a division by 400 in the NVM current will yield subthreshold magnitudes ($< 1\mu A$),

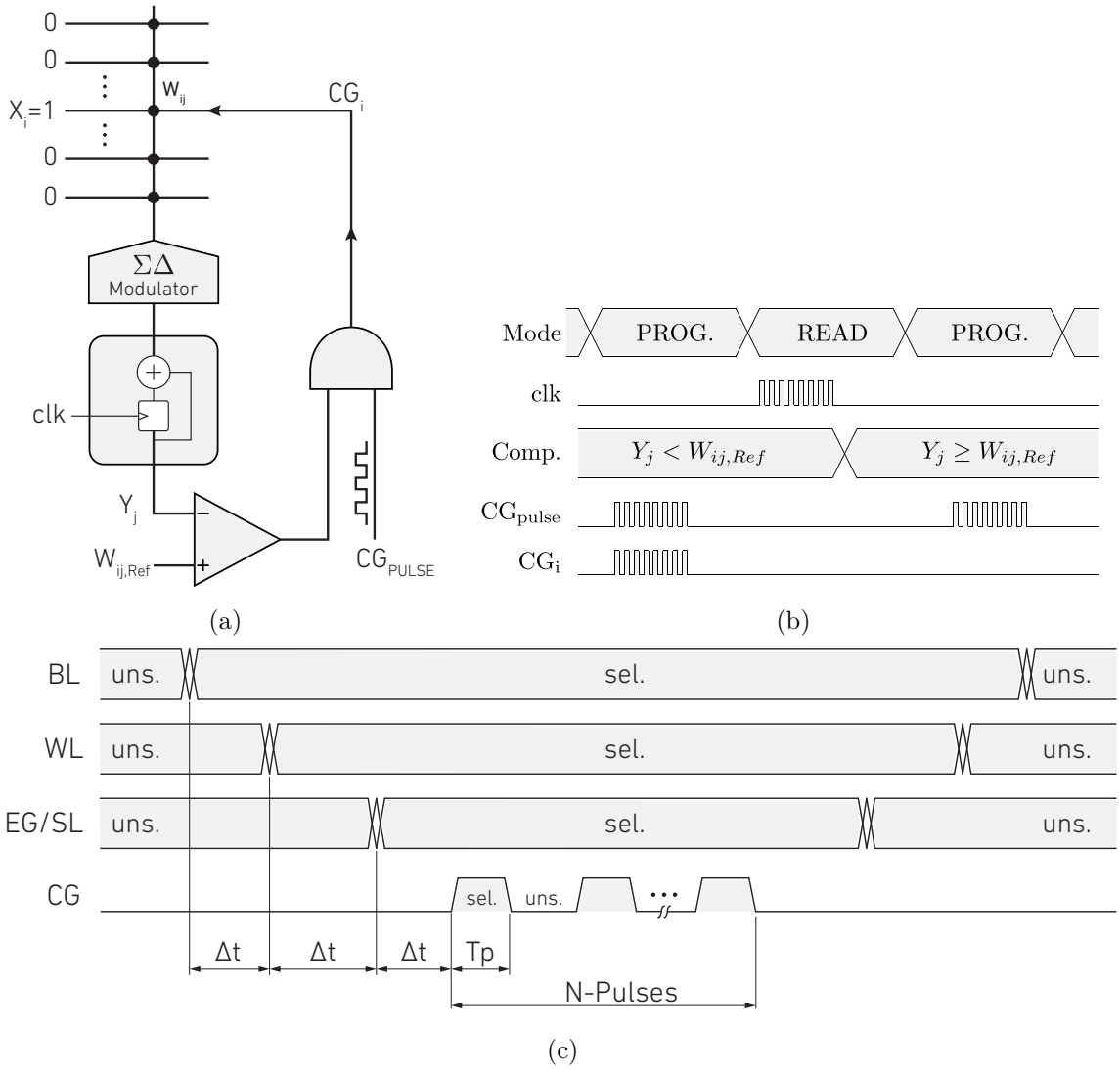


Figure 5.13 ***(a)*** *A high-level block diagram is shown for the multi-bit programming scheme for W_{ij}. After an erase, the bit-cell is iteratively programmed and read until the readout value Y_j is equal or greater than the reference $W_{ij,Ref}$.* ***(b)*** *A timing diagram for the last programming cycle.* ***(c)*** *Timing diagram for NVM control signals for program operation. The selected (sel.) and unselected (uns.) values refer to Table 3.1. Values Δt, T_p and N are parameters stored in cache. A typical program-time, T_p, is $1\mu s$.*

and noise may become a problem if it is not averaged long enough to achieve a useful precision.

5.2.3 Multilevel programming

As mentioned earlier, multilevel programming is achieved by iterating program and read operations. In this section, a more detailed, yet high level description of this

procedure is described. A block and timing diagram of the programming scheme is shown in Fig. 5.13, and will be explained below:

1. The first step in programming is to perform an *erase* operation, which sets the readout current to its maximum value ($\sim 37\mu A$).

2. Next, the NVM is set to program-mode, and either a pulse, or a train of pulses is sent to the target cell's coupling-gate (CG_i) following the operation described in Fig. 5.13c and Table 3.1. The amount of pulses as well as pulse duration are programmable using parameters stored in cache, and are reflected in the timing sequence shown in Fig. 5.13c.

3. Next, the NVM is set to read-mode and the output of the sigma-delta is accumulated over M-cycles (up to $2^{16} - 1$). The readout, Y_j, is compared to a reference, $W_{ij,Ref}$, and the result is used to mask program pulses (CG_{pulse}) during the next program cycle (this effect can be seen in Fig. 5.13b). During the read/program phase, only one row is enabled at a time, allowing only the current of the cells being programmed to be read. Noteworthy, all 512-columns can be programmed simultaneously with different values.

4. Repeat steps 2–3 until $Y_j \geq W_{ij,Ref}$.

The main advantages of using a feedback loop to program the floating gates, is that the read/program sequence naturally compensates readout offsets and shifts in the current mirrors. The idea lies in the fact that a particular cell is set to read out a specific digital value. However, most likely there will be some non-linearity when the currents of two, or more cells are added, which is why the sigma-delta accumulator includes a non-uniformity correction scheme.

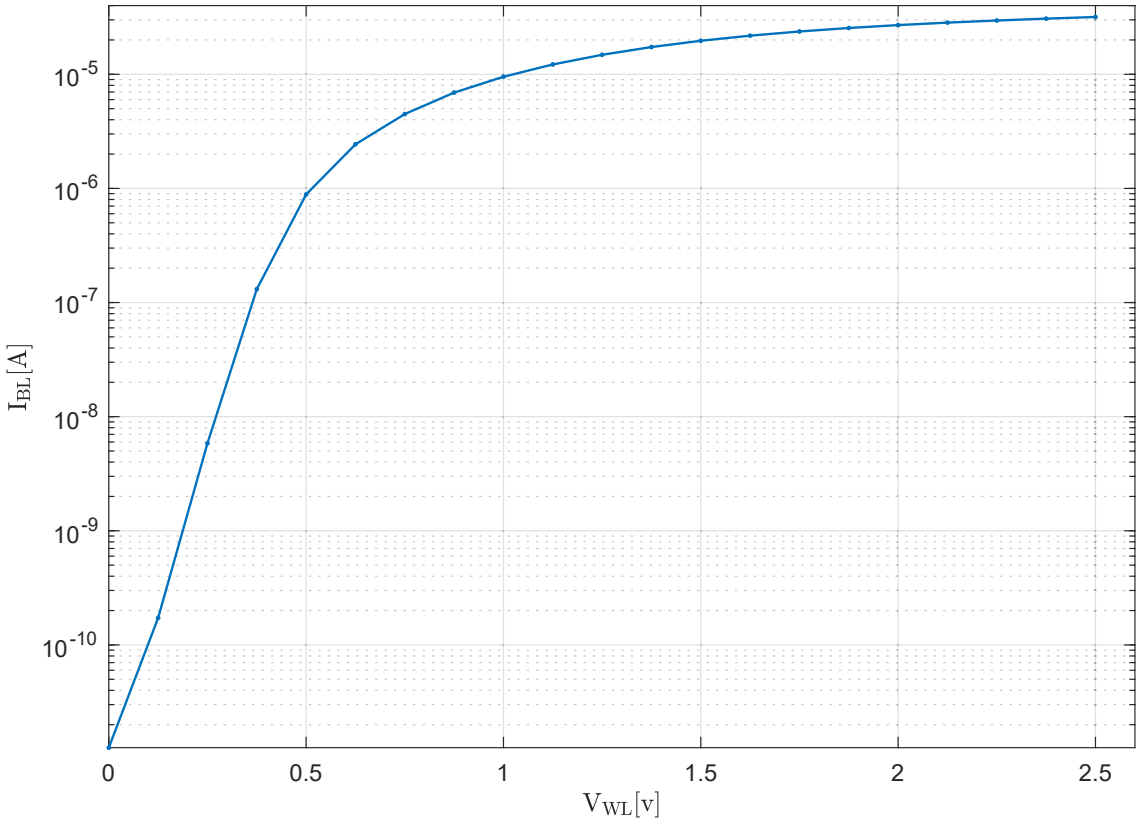


Figure 5.14 *Measured I-V curve of a fully erased floating gate, showing how the cell may operated in the subthreshold region.*

NVM Operation under Noisy Conditions

This section is dedicated to demonstrating the ability of the NVM to operate under noisy conditions, namely shot noise. This is accomplished by analyzing empirical test data, shown in Fig. 5.14, in conjunction with simulated shot noise (shown in Fig. 5.12 for a 16 clock cycle bandwidth). Data shows that by programming the cell gradually into the subthreshold region, 4-bits or more can be stored in the cell and subsequently used to carry out computations in a true mixed-signal domain.

Let us consider the total bit-line current in an NVM column as $I_j = 25nA \times 400 = 10\mu A$ (recall that 25nA is the nominal current of the A/D). The former is nodal summation of the individual cell-currents, $I_j = \sum_k i_{j,k} = 10\mu A$. As a result, the maximum current that each NVM cell can carry is 78nA, for 128-rows. Referring back to Fig. 5.12, each cell can easily store 4-bits, which is the target for this work. At

4-bits per cell, the combined current would yield an 11-bit result, out of which only the 4 MSBs will be considered for the A/D conversion.

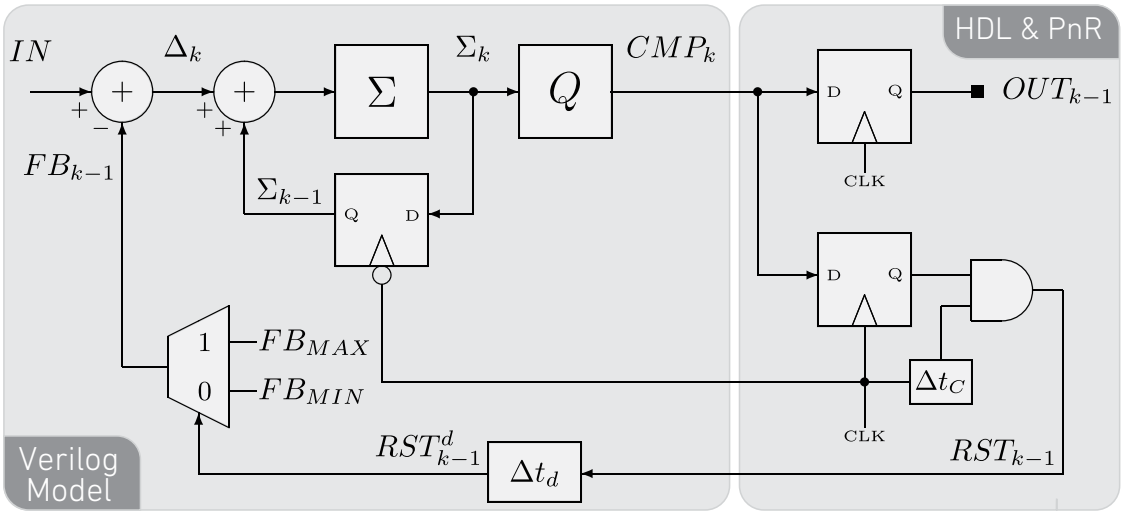
The operation described earlier is embodied as a fused-multiply-and accumulate (FMAC) [78], where the operation $a \leftarrow a + (b \times c)$ is carried out using full precision in the analog domain, and later re-quantized down to 4-bits. Finally, a row-parallel 4-bit input will take 16 compute cycles to complete, where the partial results of the dot-product are stored in the sigma-delta accumulator. Since the input-coding is unary, a full dot product would require 256 clock cycles, and would result in a quantized output of 8-bits. The full operation can be seen below (Eq.5.23):

$$z = \sum_{Acc_p} \left\lfloor \left(\sum_{Row_k} x_{p,k}\, w_k \right) \right\rceil_{N=4-bits}, \tag{5.23}$$

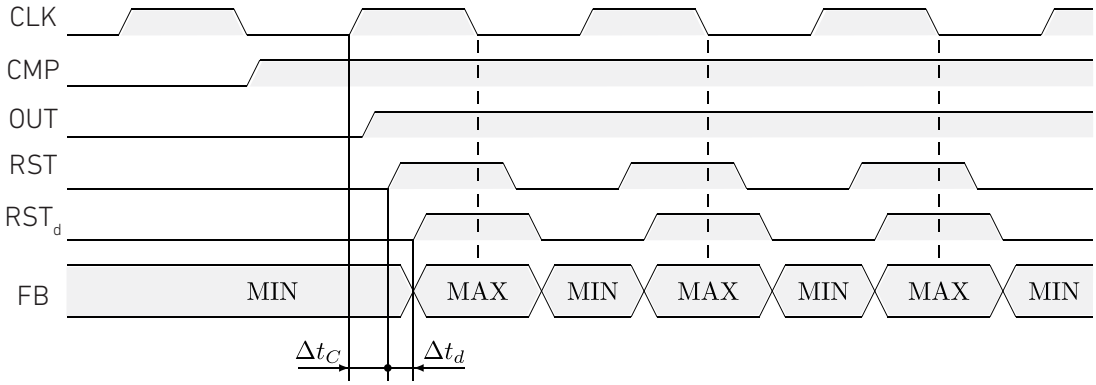
Where the $\lfloor \cdot \rceil$ operator indicates a rounding (or re-quantizing) of the result down to $N = 4 - bits$, $k = 128$ rows, and $p = 16$, refers to the aggregate accumulate time-steps across the inputs pulse-width-modulation.

5.2.4 Simulation Models for the Mixed-Signal System

System-level simulations which include analog and asynchronous circuits such as the sigma-delta converters require high simulation times ($\sim$ days), making this solution nonviable. In order to mitigate this problem, low level Verilog-HDL models were created allowing interface between digital synthesized and manual circuits. A behavioral model of the NVM array was created and described in Section 3.3.2. This model was validated via FPGA emulation and later utilized in the functional verification step of the Suanpan chiplet. The remainder of this section is dedicated to describing the model utilized for the sigma-delta A/D converter.



(a)



(b)

Figure 5.15 ***(a)*** *First order sigma-delta modulator simulation model. The digital place-and-route (P&R) section is shown on the right side, while the manual part is shown on the left. The key to the model is the feedback mechanism, which is adaptively set to the maximum input data from the NVM.* ***(b)*** *Timing diagram for the simulation model accounting for delays in the system.*

Sigma-Delta Verilog-HDL model

The first order sigma-delta modulator was explained in Section 5.2.2, and its respective HDL simulation model and timing diagram is shown in Fig. 5.15. The left side of Fig. 5.15a encompasses the manual portion of the layout, while the right side represents the interconnect to the digital synthesized circuits. The main factors considering this model are described below:

Table 5.4 *Verilog-HDL snippet for sigma-delta model*

```
parameter feedback=(2**membits-1)*ROWS;
parameter vth=(2**membits-1)*ROWS;
// 2ns delay in reset signal
assign #(2) rst_delay=rst;
// feedback decision block
always @* begin
  if (rst_delay==1)
    fb=feedback;
  else
    fb=0;
end
//Sampled process on negative edge
always @(negedge clk) begin
  if (model_reset==1) begin
    delta=0;
    sigma=0;
  end else begin
    // summing junction
      delta=in-fb;
    // integrator
      sigma=delta + sigma;
  end
end
// quantizer logic
always @* begin
  if (sigma>=vth) begin
    ncmp_int=0; // this is not(cmp)
  end else begin
    ncmp_int=1; // this is not(cmp)
  end
end
assign ncmp=ncmp_int;
```

- The internal accumulator uses a negative edge triggered register, allowing a significant time delay for the generation of the feedback control signal in the synthesized digital block. This is particularly useful when simulating post-layout back-annotated Verilog netlists during the functional verification step.

- The feedback is 1-bit D/A, where the maximum feedback (FB_{MAX}) is set to the maximum input from the NVM readout, and the minimum feedback is set zero. The D/A is modeled with a multiplexer, yielding a discretized output with only two outcomes. The maximum feedback value depends on the number of bits used to model the NVM cell and the number of rows in the NVM array. A

snippet of the Verilog code is shown in Table 5.4.

In the digital domain, a first order sigma-delta modulator acts as a re-sampling device, where the input data is re-scaled to some predefined range, which is set by the maximum feedback value. Recalling, when the desired output is a consecutive stream of ones, the feedback is set to maximum, equating to the input to the A/D converter. In the case of the analog A/D, the feedback is chopped by the clock, therefore, the feedback-current is tuned to be approximately twice of the maximum input current.

5.2.5 Analog-Digital Interface

Table E.1 shows the interface ports between the analog and the digital portion of the system, and Table E.2 shows the necessary power supplies and biases to operation the mixed-signal portion of the core.

5.2.6 Processing Unit Configuration

The design utilizes input, control, and accumulator registers to store parameters important to current computations. In addition, the **xmask** and **y-mask** (column-mask) registers provide the array with the flexibility to target single cells or subsets, facilitating kernel processing and convolutions in shift register operations. The data encoder, control unit, and accumulator will be described in more detail below. The digital core operates as follows:

- **CONFIGURE PU:** The control and accumulator configuration registers are reset and loaded with the desired parameters via the network interface. Through the decoder, every memory element can be targeted via the *write enable* and *write address*. Identical operations are conducted on the row and column *mask* which identify the cells to process. Then, the accumulator parameters (offset and increment) are loaded in. Finally, the input values are loaded into the

128x8 register file that is read out in parallel to the encoder. To note, data comes into the PU from the network in 256-bit packets. In order to load and configure registers, multiple packets are needed. The worst-case scenario is loading the accumulator parameters which requires 32 addresses. Therefore, the bottom 5 bits of the write address are reserved for the addressing the various elements. The next 3 bits are used for decoding which element is being written to. Table E.3 shows how the decoding was accomplished.

- **ERASE:** After all configuration registers have been loaded, an ERASE operation is executed. Using the *erase address register* loaded from the network, the HV switches are configured and held in *erase state* for a parametrizable amount of time, based on the values of the *control register*.

- **PROGRAM CELLS:** Following a comprehensive ERASE, rows can be targeted for programming. As described in a prior section, these cells have the capability to be programmed at multiple levels which is specified by a value stored in each columns' 8-bit *offset register*. The programming algorithm is shown below:

    1. The length and number of programming pulses are set by the *control register*. The programming pulse is wired to one of the bits of a 32-bit counter and is re-iterated up to 4096 (12 bits) pulses.
    2. Using the *program count register*, each pulse is separated up for a flexible amount of time, progressing from BL to WL, to SL/EG and finally to CG (as shown in Figure. 5.15.
    3. The programming pulses are used to multiplex (mask) the CG signal.
    4. After an initial program, the cells are read for a desired number of bits defined by the *end count register* (2^N bits).

5. At the end of a read, the value is compared to the *offset register* and it is determined whether the operation has completed.

6. These steps are repeated until every cell has converged to its desired programming value (unless a one-time program is specified, shown in Table E.6).

- **COMPUTE:** After programming specific rows, the COMPUTE operation is conducted. Using the *control register* to specify the desired number of bits on the input and the *end count register* to specify the number of bits on the output, the vector-matrix product is calculated. 2^P successive reads are accumulated to produce the result.

- **ADJUST PARAMETERS:** After reading out the results of a COMPUTE operation, parameters can be adjusted such as input, output bit resolution, accumulator increment and offsets, and the mask values. Additionally, the row mask, which is implemented as a barrel shifter, can be adjusted by a desired amount specified in the control register (up to 127 bits) which encourages and allows kernel processing.

- **RECOMPUTE:** Following any re-adjustment, the processor can re-compute the product once again and the subsequent result is read out into the network in 256 packets which encompass 16 columns (each 16-bits). Therefore, to read out all 512 columns, 32 READ requests are needed for the NVM PU. Because of this, the bottom 5 bits of the read address are reserved for addressing the output of registers while the next three bits are used to multiplex which component's output is being sent to the network.

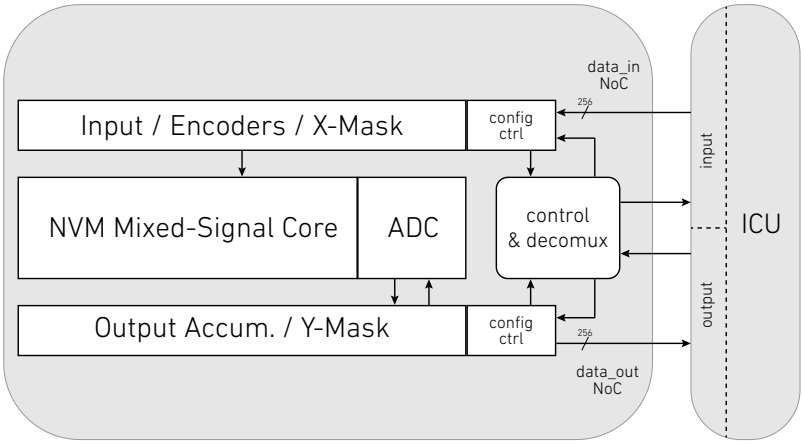


Figure 5.16 *High level block diagram of Suanpan system and its main peripherals.*

Data Encoder

The encoder comprises a pulse-width-modulation scheme, where all inputs (stored in their respective registers) are compared with a digital ramp. Through the control unit, the width of this PWM signal can vary from 2 (1 bit) to 256 (8 bit) clock cycles which quantizes the input data into the desired number of bits. As mentioned earlier, the array has three modes of operation: erase, program, and read. During a compute operation, the encoder is enabled for 2^P cycles while for program and erase the encoder is essentially idle. Table E.4 shows how the encoder is configured via the *control register* and how it relates to the input bit resolution.

Control Unit

The control unit is the brain of the NVM PU as it controls the important, peripheral digital blocks and sends the configuration signals to the analog switches. This block is configured by auxiliary control registers seen in Fig. 5.16 which are loaded from the interface and detailed more closely in the Table E.5.

Since there are three operations of the NVM PU, the control word can be configured in three unique ways as well. To note, during program, a one-time program functionality was implemented as well for debugging purposes. Table E.6 shows how the control

register was implemented bitwise.

The unit has direct control over the *accumulators*, *encoder*, and *xmask*. For the accumulators, which will be further covered in the subsequent section, the unit can either reset the array to the offset value or zero, enable counting (with flexible increment), or hold the array in idle. In addition to controlling the accumulator, the control unit determines if there is a subtraction operation required after a read. The encoder's length of operation is controlled by the unit by its changing of the ramp counter's increment from 1 to 128. Finally, the xmask register is shifted by a value from 0-127 (length of the register) at the beginning of a compute operation depending on the value of the control word.

In addition to its control over digital blocks, the control unit mostly functions as a mechanism to set the configuration signals for the analog switches, which ultimately determine the operation of the NVM array. These control bits are configured and timed in the manner addressed in the previous section, detailing the analog switches. To note, the logical AND between the xmask-register and the output of the encoder is used to multiplex the switch input to the analog block, where '0', indicates a word-line ground state. Conversely, a '1' makes the switch adhere to the control block configuration. Likewise, the column mask multiplexes the switch input to the bit-line. The other three ports (EG, CG, SL) are taken verbatim from the control block.

Accumulator

A block diagram of the accumulator is shown below in Figure. 5.17. The accumulator has four states: program reset, compute reset, count, and idle. When reset, the accumulator arrives at the offset value or at zero, depending on the desired control operation. It is important to reset to zero instead of the offset value during program, because the desired program value for a specific cell is stored in the *offset register*. During the count state, the output of the $\Sigma\Delta$ modulator determines whether the

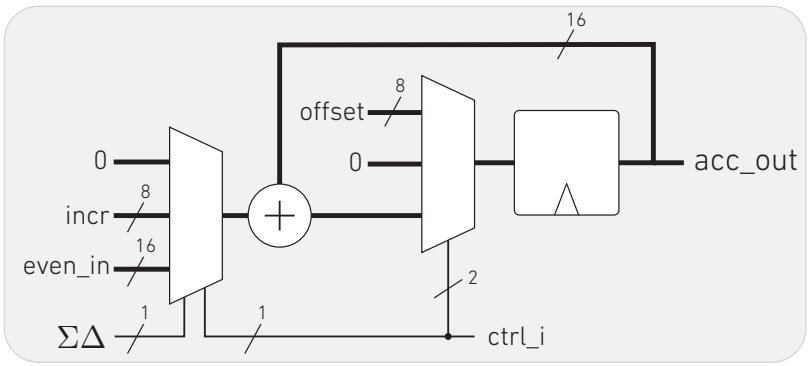


Figure 5.17 *Accumulator architecture*

accumulator will increment or not. At the end of a read, the value accumulated by even columns is subtracted to the that of their odd counterparts. For a program operation, the increment is set with respect to the end count as seen in Table E.7. In order to signal when a cell has been programmed, the upper 8 bits of the accumulator are compared to the offset register.

5.3 Results

Figure 5.18 shows the top-level layout of the mixed-signal Suanpan core. The blocks outlined in orange were laid out manually, and the rest was placed & routed using Cadence Innovus EDA tools. The figure shows all internal blocks and their approximate footprint. The PU footprint was designed such that it could be replaced with other tiles, or vice-verse. Table 5.5 shows the summary of the design specs for the mixed-signal NVM computational unit. The Suanpan tablet has been fabricated twice and is currently in preparation for testing. The first iteration resulted in inconclusive testing due to issues in the I/O interconnect. Specifically, the I/O pads had a bug in the internal connectivity that the verifier did not catch.

Core Efficiency

Table 5.5 *Design specs for the NVM mixed-signal computational unit.*

Technology	55nm GlobalFoundries LPX
Core area	0.176mm^2
Core voltage	1.2v
Transistors	180K
Array dimensions	128×512
NVM memory	256KB
Bit-Cell area	0.17μm^2
Input precision	4-bits
Parameter precision	4-bits
Output precision	8-bits
Operation	Fused Multiply-and-Accumulate (MAC)
Clock frequency	300 MHz
Clock cycles per operation	256
Operations per second	76GMACs/s
Energy Efficiency	107TMACs/Watt

In order to calculate throughput, first the operation, MAC, needs to be defined.

In this work, the actual operation can be viewed as a fused-multiply-and accumulate (FMAC) [78], where $a \leftarrow a + (b \times c)$ is carried out using full precision in the analog domain, and later re-quantized down to 8-bits. A row-parallel 4-bit input will take 16 compute cycles to complete, where the partial results of the dot-product are stored in the sigma-delta accumulator. Since the input-coding is unary, a full dot product would require 256 clock cycles, and will result in a quantized output of 8-bits (see Eq.5.23). This result is evaluated in $Tc = 256/300MHz = 0.853\mu s$, and equates to a total of 128 MACs. Following this train of thought, the array throughput is calculated as $512 \times 128MACs/0.853\mu s = 76GMACs/s$.

The energy efficiency is defined as the number of operations per watt (MACs/Watt). The energy required to complete a compute cycle is the same as computing 16-fold 4-bit A/D conversions and accumulating the result. Following Table 5.3, the total A/D conversion energy computes to $E_{\Sigma\Delta} = 1.184pJ$. Additionally, 1 MAC may be computed in 6.66ns, which gives $1/6.66ns = 150MMACs/s$, and an average power of $1.184pJ/0.853\mu s = 1.39\mu W$. Finally, $E_{ff} = 150MMACs/1.39\mu W \sim 107TMACs/Watt$.

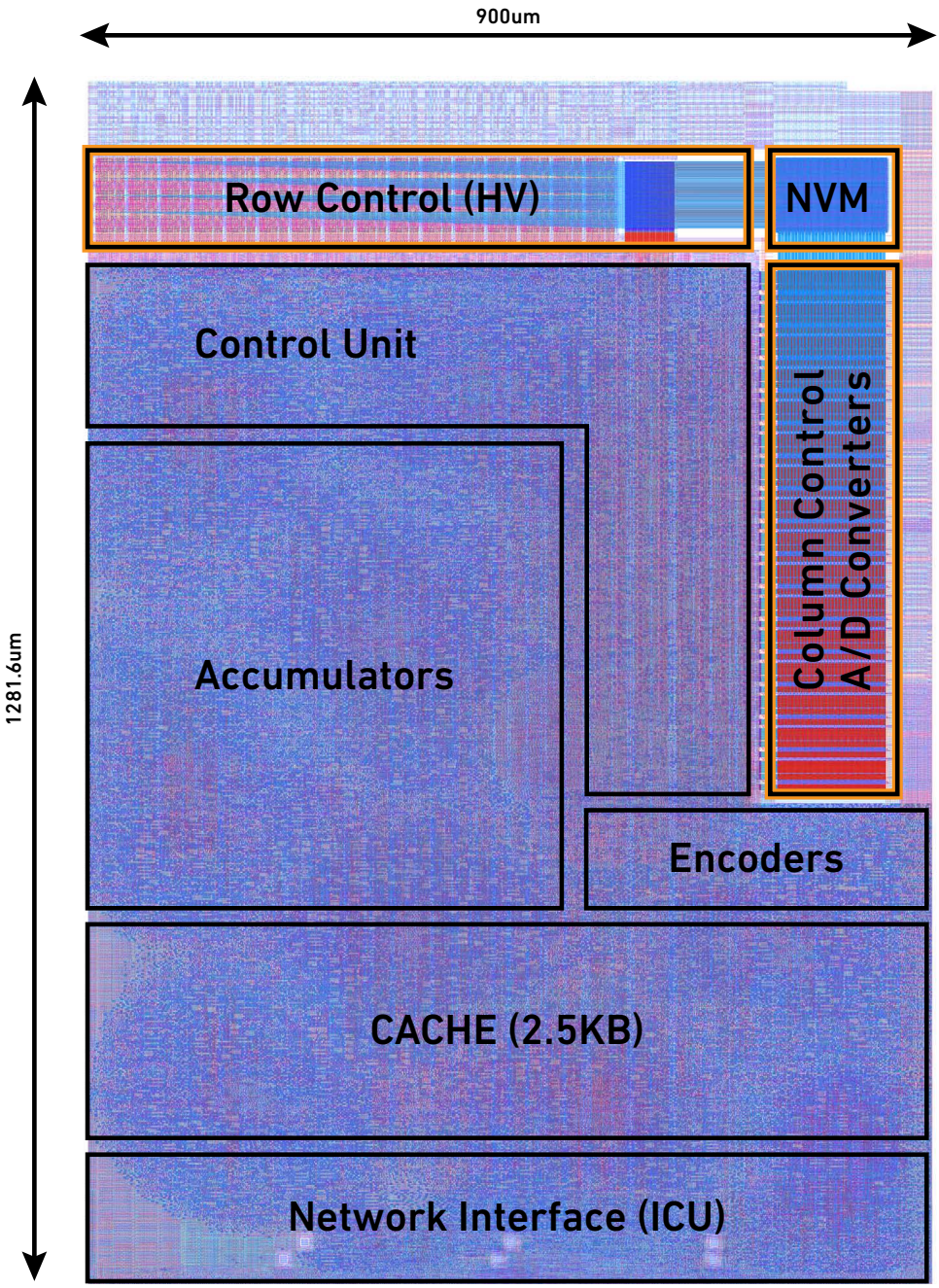


Figure 5.18 *Top level layout of the NVM Suanpan mixed-signal core.*

5.4 Conclusion

A System-in-Package (SIP) edge processor was designed as part of a group effort with the goal of providing a high energy efficiency and computational bandwidth for an image processing pipeline. The SIP's substrate consists of a passive silicon interposer fabricated in a 1μ technology, which houses three Chip-Multi-Processors (CMPs), a 3D DiRAM memory module, and a Zynq 7 FPGA. The CMPs are comprised of heterogeneous processing cores which are in charge of providing real-time hardware acceleration of image processing tasks, such as DeBayering, dewarping, background segmentation, pattern matching, and any custom Machine Learning (ML) algorithm that requires high-speed fixed precision vector processing.

In the scope of this thesis, a Compute-in-Memory processor is described, the *Suanpan* tablet. Suanpan was designed and fabricated in a CMOS 55nm technology with the purpose of accelerating vector-matrix multiplications (VMMs). The mixed signal tile was instantiated on a 2D Network-on-Chip, thus allowing communication to/from main memory (DiRAM) via a token ring, as well as neighboring processors and co-processors via a mesh-grid switch packet network.

Suanpan's architecture is composed of a 128×512 NVM computational crossbar, equipped with row and column high-voltage control and addition to a column-parallel bank of first order sigma-delta A/D converters. The crossbar was designed to support multilevel storage, and A/D converters were avoided in the input by using time-domain PWM encoders. The analog current readout provides low power and massive parallelism. A custom processor provides direct-memory-access control for external bus masters an allows for digital post-processing to take place locally in the core.

Suanpan was fabricated twice and is in preparation for testing. The first test iteration resulted inconclusive due to a wiring bug in an I/O pad cell; after fixing the issues, the chip was re-fabricated and bonded. Simulation results show that the NVM

array can store 256KB of data, producing 76GMACs/sec and 107TMACs/W, and allows for both binary and unary multilevel storage.

Chapter 6

Compute-in-Memory (CiM) Processor Array for Artificial Intelligence Edge-Computing

6.1 Introduction

In-memory computing addresses the communications bottleneck that conventional architectures face. This is accomplished by embedding ubiquitous arithmetic operations such as Multiply-and-Accumulates (MACs) within compact memory cells. Chapter 4 showed the potential of DRAM primitives in CiM accelerators and as a result, matrix parameters were stored locally thus reducing the energetic cost of fetching data either off-chip or in neighboring memories. Charge-based computing is attractive due to its conservation properties and ability to operate near thermal-noise limits without losing precision, and low voltage swings ($\sim$ 100mV) on integration capacitances allow even further reduction of energy. Additionally, the DRAM memory designed in this work does not require special process options (such as those required for non-volatile memory), which makes it the most attractive design for integration in logic processes.

Chapter 5 addressed the integration challenges that high-throughput machine-learning (ML) accelerators pose, which combine dedicated memory controllers, and massive parallelism for computation. As a result, a new system-on-chip (SoC) multipro-

cessor was designed by combining the two aforementioned concepts and implemented in a CMOS 65nm node. The project was a result of a one-year duration collaboration with Northrop Grumman Corporation. The design of computing cores was carried out by the Johns Hopkins Team, and physical implementation was completed by engineers at Northrop Grumman Corporation.

In this work, compute modularity and parallelism are achieved by replicating identical mixed-signal CiM units, and flexible programmability is achieved by embedding RISC processors next to the CiMs. The first part of this chapter describes system-level implementation details and exposes the SoC architecture, which stresses the use a Network-on-Chip (NoC). In the second part of this chapter, one Core Unit (CU) is described, which contains CiM units which can carry out vector-matrix multiplications. Analog and digital circuits are described, and finally, results and conclusion are presented.

6.2 System-on-Chip (SoC) Architecture

The System-on-Chip architecture described in this chapter is shown in Fig. 6.1 and consists of a High-Speed-Interface (HSI), 2 SiFive CPUs, and 56 compute-slots (7 × 8). The former houses 6 cache units (CU_AUX), 1 is General Purpose Input Output (GPIO) unit, and 49 Core-Units, which in turn contain 2 Compute-In-Memory (CiM) cores within. All designs are arranged on a 2-layer Network-on-Chip (Noc) and are allowed to communicate with each other, and with off-chip memory.

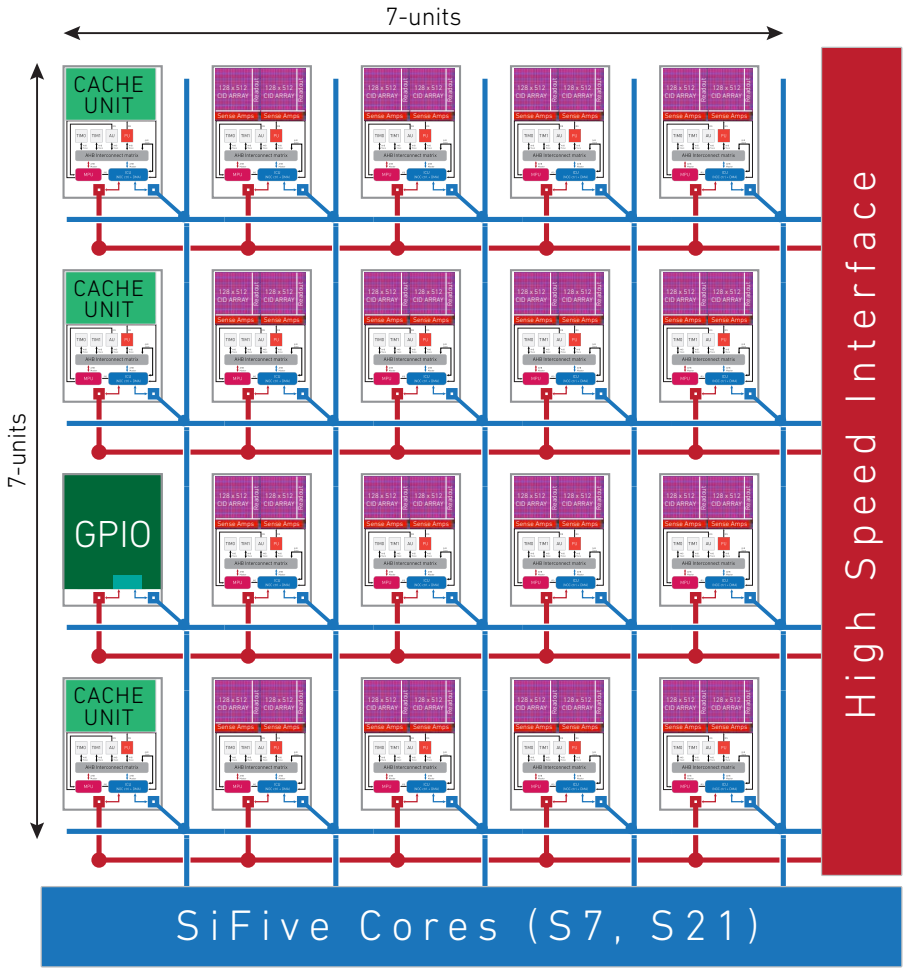


Figure 6.1 *System-on-Chip block diagram consisting of 49 cores, cache units, two RISCV SiFive processors, a GPIO interface and a High-Speed Interface (HSI). The cores rest on a two-layer Network-on-Chip and are free to receive data from any peripheral in the system.*

As mentioned earlier, each Core-Unit (CU) encompasses 2 mixed signal CID vector-matrix multiplier cores (CiM Cores) in addition to a custom co-processor and a network interface unit. A High-Speed Interface (HSI) is used to stream data onto a token ring network (N1), while a GPIO interface is used for pushing data onto a mesh grid network (N2) utilized for diagnosing and configuring chip parameters. Moreover, several cores along the left-most column are utilized as cache units within the same infrastructure, thus enabling neighboring cores to access data stored locally on chip. Additionally, two RISCV SiFive processors are used to bootstrap the system and coordinate the work of the numerous processing units as well as facilitate other computations when convenient. The cache units contain a network controller and SRAM ($\sim$ 100KiB), which is intended for on-chip storage.

The remaining sections in this chapter focus on the design and implementation of the mixed-signal cores as well as the system that contains them. The cache-unit, HSI, and SiFive Cores fall out of the scope of this thesis.

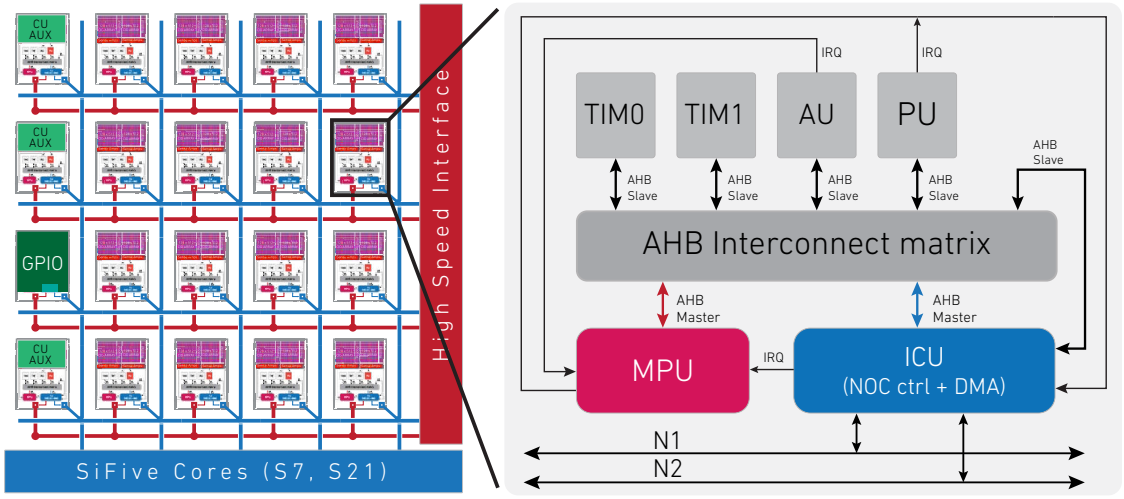


Figure 6.2 *Block diagram of the Core Unit (CU), which encompasses the following blocks: Main Processing Unit (MPU), an Interface Control Unit (ICU), an AHB interconnect matrix, and several AHB Slaves. The AHB slave ports contain Tightly Integrated Memory (TIM), Auxiliary Units (AU), and the Processing Unit (PU).*

6.2.1 Core Unit (CU) Architecture

The Core Unit (CU) constitutes a single computational node which lies on the aforementioned two layer NoC (shown in Fig. 6.1). There are 49 CUs in total, which in turn encompass the following blocks, also illustrated in Fig. 6.2:

- **Processing Unit (PU):** This block is composed of one or more CiM Cores and a configuration unit (ESC Conf.). The CiM cores are part of the Edge-Super-Computing (ESC) Subsystem, explained in further detail in Section 6.2.2. The PU is programmed as bus slave which allows offloading of intensive vector processing tasks, thus improving overall system computational performance. Additionally, it uses its DRAM array as auxiliary memory, which may be accessed by any bus master. The ESC Core (CiM Core) is controlled indirectly via a custom instruction-set-architecture (ISA) processor, which is in charge of communicating with any bus master and interpreting opcodes that are received. Similarly, the configuration unit (ESC Conf.) is also a bus slave and is used to configure all parameters relevant to the PU, such as compute precision.

- **Main Processing Unit (MPU):** This block is a custom RISC instruction set processor is dedicated to supervising dataflow and pre/post processing any data to/from the PU. The MPU is a bus master and its design of the MPU falls out of the scope of this thesis.

- **Tightly Integrated Memory (TIM):** Comprised of block SRAM, the TIM is a passive bus slave and is used to store local data and parameters relevant to both MPU and PU.

- **Interface Control Unit (ICU):** This block is used to address the communication between the CU and both layers of the NoC (N1 and N2). Additionally, the ICU incorporates a direct-memory-access (DMA), which oversees memory

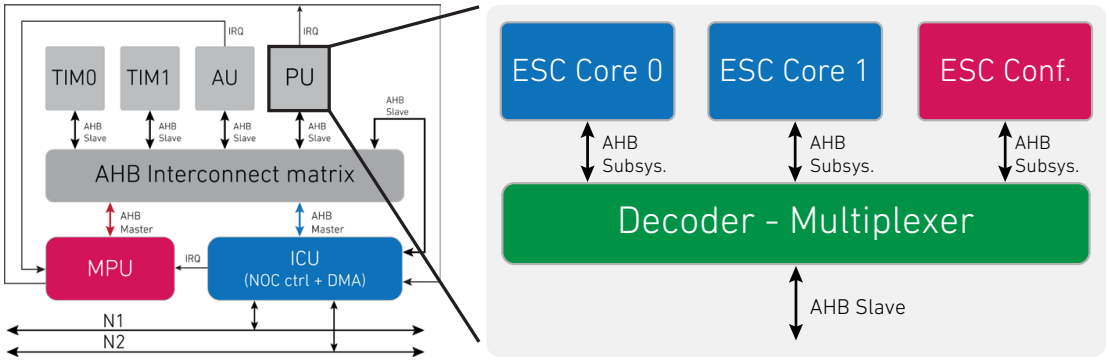


Figure 6.3 *Processing Unit (PU), also referred to as the ESC Subsystem, which is the main computational block of the ASIC system. All AHB Subsystems are connected via a decoder-multiplexer according to the AHB protocol.*

intensive tasks. This allows the MPU to be used simultaneously in auxiliary processing tasks. The DMA may be programmed by the MPU or the NoC.

- **AMBA 3 AHB Lite interconnect matrix:** the AHB matrix allows several bus masters (mainly MPU and NoC) to communicate with the various slaves, providing adequate arbitration. The AHB matrix description falls out of the scope of this thesis.

Since the system operates on an AHB Matrix interconnect, all blocks and sub-blocks are mapped in memory and can be addressed from both off-chip or on-chip. In particular, the Processing Unit contains its own AHB memory subsystem, shown in in Fig. 6.3, where a decoder-multiplexer allows any bus master to write/read data to/from any register in the slave.

6.2.2 Processing Unit (PU) - ESC Subsystem

The Processing Unit (PU) will also be referred to as the Edge-Super-Computing Subsystem (ESC Subsystem). The ESC Subsystem lies within a Core-Unit (see Fig. 6.2) and is composed of two *ESC Cores* (also referred to as the CiM Cores) and a configuration unit (*ESC Conf.*).

ESC Core (CiM Core)

The goal of ESC Core is twofold:

- Its main purpose is to serve as mixed signal edge computing vector processor; as an edge device, it is programmed on-demand by any bus master. For example, it is given a set of instructions (opcodes) via the AMBA bus, and subsequently interacts with a *master* via interrupts (or a polling mechanism). Opcodes are interpreted by a dedicated controller (see Section 6.3.4), which generates all necessary signals for stimulating the analog compute blocks. Upon completion of a compute task, the ESC core sets an internal status register high; it is wired to the output of the slave and can be used by any bus master. For example, the MPU can either poll the register or use it as in interrupt per se. This allows any bus master to carry out ad hoc tasks in parallel without having to implement out-of-order instructions.

- Its secondary purpose is to act as a passive memory slave, which may be accessed by any bus master in the system. A dedicated controller takes care of interpreting opcodes, which are used to indicate whether a read or write is requested. The access protocol is asynchronous in that the controller utilizes interrupts to communicate when an operation has been completed.

The ESC Subsystem's base memory location is shown in Table F.1, and Table F.2 describes the memory space therein. The former lays in a multi-layer AMBA 3 AHB-Lite interconnect, whose top-level block diagram is depicted in Fig. 6.3. Since the system is built on an AHB Bus, a PU may contain an arbitrary number of accelerator cores, which are limited by die area.

6.3 A Fully Programmable Mixed-Signal Pseudo-DRAM CiM Processor

A mixed signal pseudo-DRAM CiM processor was designed and implemented as part of the SoC processing array described in the previous section. The CiM processor will is referred to as the Edge-Super-Computing Core (ESC Core) and is the focus of this chapter.

The ESC Core is responsible for accelerating intensive vector processing tasks that may be offloaded by any bus master. The core lives inside a Processing Unit and connects to it via the interface described in Table F.3. It consists of a digital-input, analog-compute, digital-output block, where The D/A conversion is accomplished directly via unary-charging of internal nodes. Charge Injection Device (CID) bit-cells are used to compute 1-bit AND operations between an external input stored data in DRAM. The analog-compute is embodied as a change in the internal charge of the circuit, which in turn is integrated in a capacitor as a voltage, and later converted to digital via a custom ADC.

The core is programmed entirely via AMBA 3 AHB-Lite, allowing the rest of the blocks to be controlled by internal circuitry. This provides for the subsystem to be portable, on a standard bus, and making it plug-and-play for accelerating data intensive tasks in any embedded low-power microcontroller. Figure 6.4 shows a diagram of the core and its main blocks. The black lines indicate the connection between an AMBA bus master and several digital peripherals. The interconnect of all peripherals is achieved via a decoder-multiplexer, which is not shown in the figure.

In the following sections, the ESC Core will be shown in a bottom-up fashion, starting from analog circuits and moving up to the digital peripherals and controller.

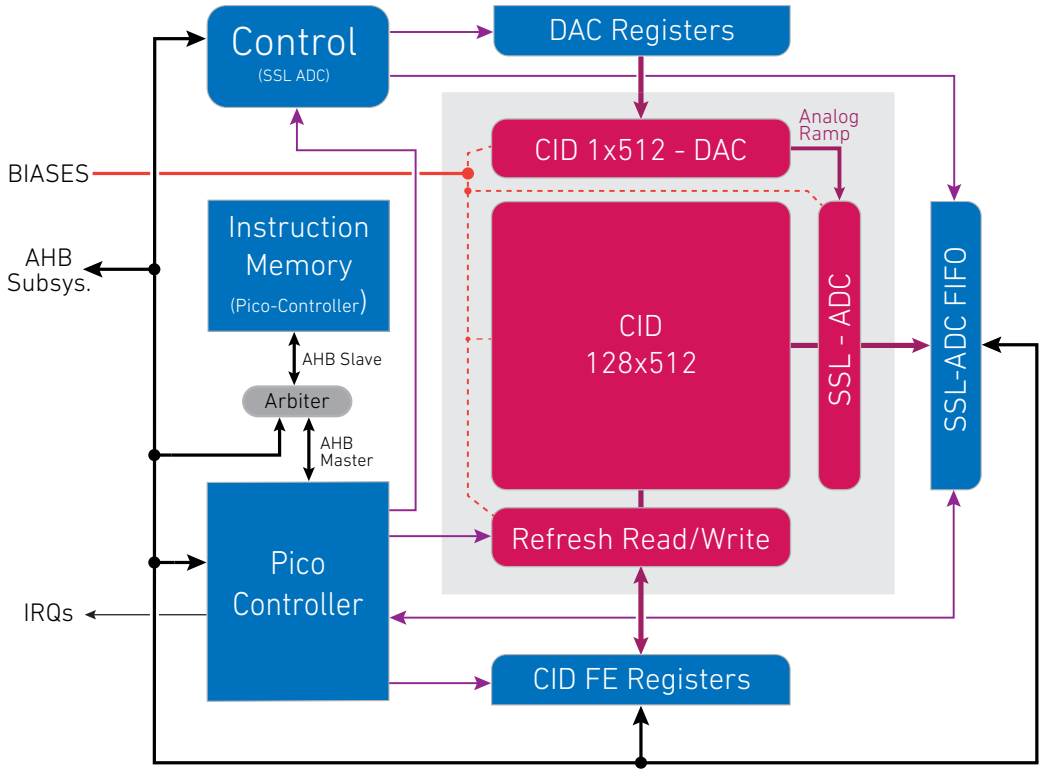
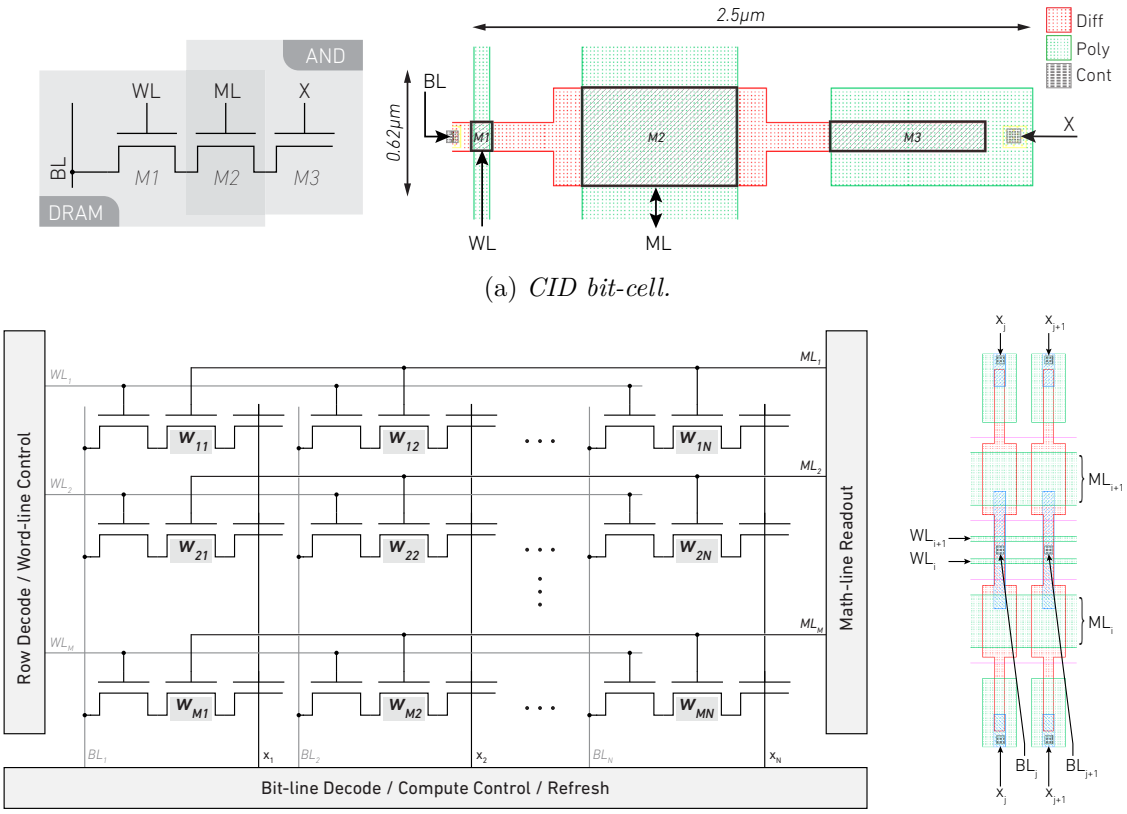


Figure 6.4 *ESC Core block diagram. Blue blocks live in the digital domain while red-pink blocks live in the analog-mixed signal domain. All registers in the system are memory mapped and communicate via AHB protocol. The top-level interface of the ESC Core Unit consists of a set of analog biases, an AHB interface and a set of interrupt signals that are mean to be wired to the MPU/DMA block.*

6.3.1 Charge Injection Device (CID) Array

In the scope of this project, a Charge Injection Device (CID) consists of a 2.5 transistor Pseudo-DRAM cell, such as illustrated in Fig. 6.5a. The left portion of the figure shows a schematic diagram of the bit-cell, and the right hand side shows the layout; diffusion is drawn in red and polysilicon is drawn in green. Access transistor *M1* is used to read/write the DRAM; charge under the gate of *M2* represents stored data, w, where negative charge represents a logic *1*. A 1-bit multiplication is performed between w and the data, x, on the gate of *M3*, thus yielding $z = w\,x$. The result of the 1-bit multiplication is reflected as a voltage change on the floating gate of *M3*, such as explained in 4.



(a) *CID bit-cell.*

(b) *CID Crossbar.*

(c) *2x2 layout.*

Figure 6.5 ***(a)*** *CID bit-cell schematic and layout showing all connections.* ***(b)*** *CID crossbar showing the common vertical and horizontal shared connections.* ***(c)*** *CID 2x2 layout showing all connections.*

Several CID cells are combined to form a 2D array, such as depicted in Figure 6.5b. The array is constructed by sharing Math-Lines along rows and inputs along columns, thus forming a compute-in-memory structure. A screen capture of a portion of array's layout is shown in Fig. 6.5c, where black arrows indicate the direction of the data.

6.3.2 Charge Injection Device (CID) Operation

Dot products between vector $\mathbf{x}$ and parameters $\mathbf{w^T}$ are carried out in the analog domain by sharing charge on the parasitic capacitance of the Math-Line. The behavior of the circuit is explained in detail in Chapter 4, with the only difference being that the $\mathbf{x}$ inputs are routed separate from the bit-line, thus reducing the storage interference through compute transistor, $M3$, in non-targeted rows. Figure 6.6 shows the CID

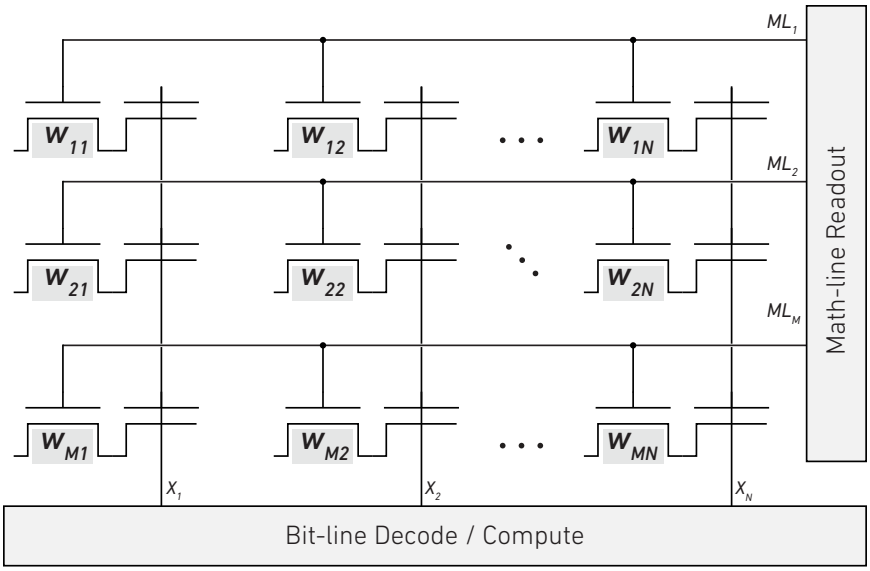


Figure 6.6 *Equivalent circuit for computation.*

array equivalent compute circuit. Access transistors ($M1$) or turned off and gates compute transistors, $M3$ are wired vertically to the **x** inputs. Compute switches are used to feed data to the inputs, and readout circuitry is used to buffer sensed charge to A/D converters.

DRAM-CID Read/Write

Figure 6.7 shows the equivalent circuit used for reading and writing to and from the array. As it can be seen, transistor $M3$ is turned off and does not participate in the operation. Likewise, the *prech* switch remains closed, thus pre-charging the math-line to $Vp + V_{offset}$. The read/write sequence is carried out entirely by using the sense amplifier, which is seen towards the bottom of the figure.

The sense amplifier provides connectivity to the bit-line (BL) via two cross-coupled inverters which are controlled by two switches: *restore* and *eqn*. Reading the contents of the DRAM array entails "opening a row", thus allowing the DRAM cells to share their charge with the parasitic capacitance of the bit-lines. The sense amplifier senses a change in voltage on the bit-line, amplifies it, and restores the logic value back to the bit-cell. Similarly, writing to the DRAM array involves "opening a row" and

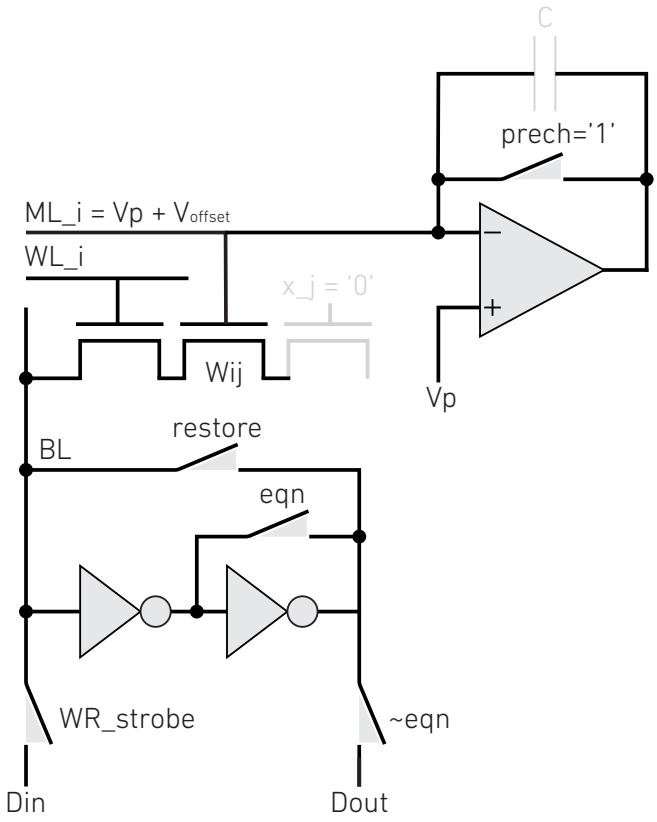


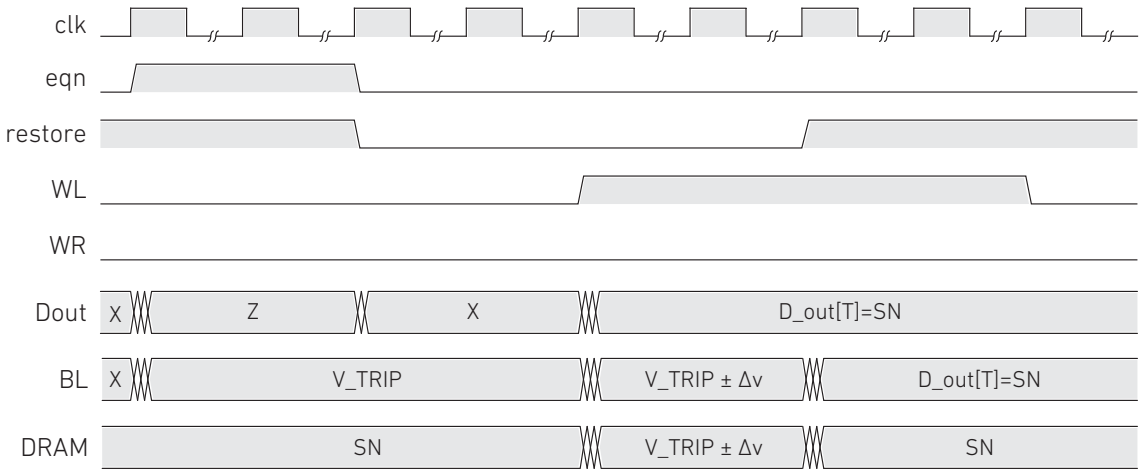
Figure 6.7 *Equivalent circuits for CID read/write and compute operations.*

forcing the content of the bit-cells to the desired write-data. Since both read and write operations utilize the sense amplifier; the only difference is that a *write* will overhaul any attempt of the circuit to perform a refresh. In general terms, the *refresh* and the *read* are identical operations; however, a *read* is carried out on-demand, as opposed to the *refresh*, where the array is usually scanned at fixed time intervals.

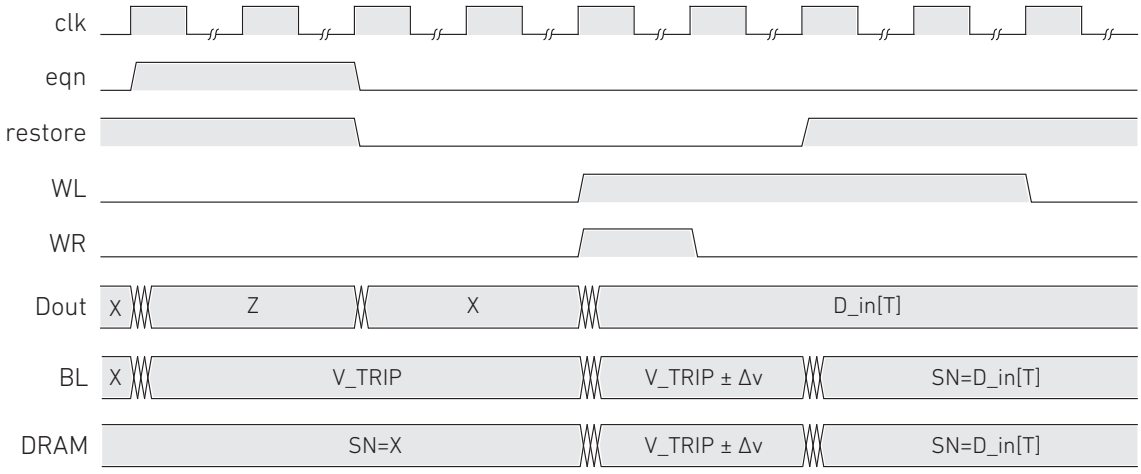
A timing diagram for the read operation is shown in Fig. 6.8a, and a simulation of two adjacent rows with different stored data is shown in Fig. 6.9. A detailed description of the refresh procedure is listed below:

- **Equalize-1:** Before starting a write/read sequence, the sense amplifiers are "equalized" by closing the *eqn* and *restore* switches. This drives the bit-line to the trip-point of the sense amplifier.

- **Equalize-2:** After the bit-line has settled to the trip point of the cross-couple



(a) *Timing diagram for reading and refreshing parameters in the CID array.*



(b) *Timing diagram for writing parameters to the CID array.*

Figure 6.8 *Timing diagrams for reading, refreshing, and writing parameters to the CID array.*

inverters, both switches are opened, leaving the bit-line un-driven, and floating at approximately $Vdd/2$ (550mV from Fig. 6.9).

- **Sense-1:** Next, the *WL* for a particular row is turned on, allowing the stored charge in the DRAM-cell to be shared with the parasitic capacitance of the bit-line. Consequently, the bit-line will experience a small voltage shift towards the logic value of the stored data.

- **Sense-2:** The sense amplifier promptly amplifies this perturbation in the bit-line,

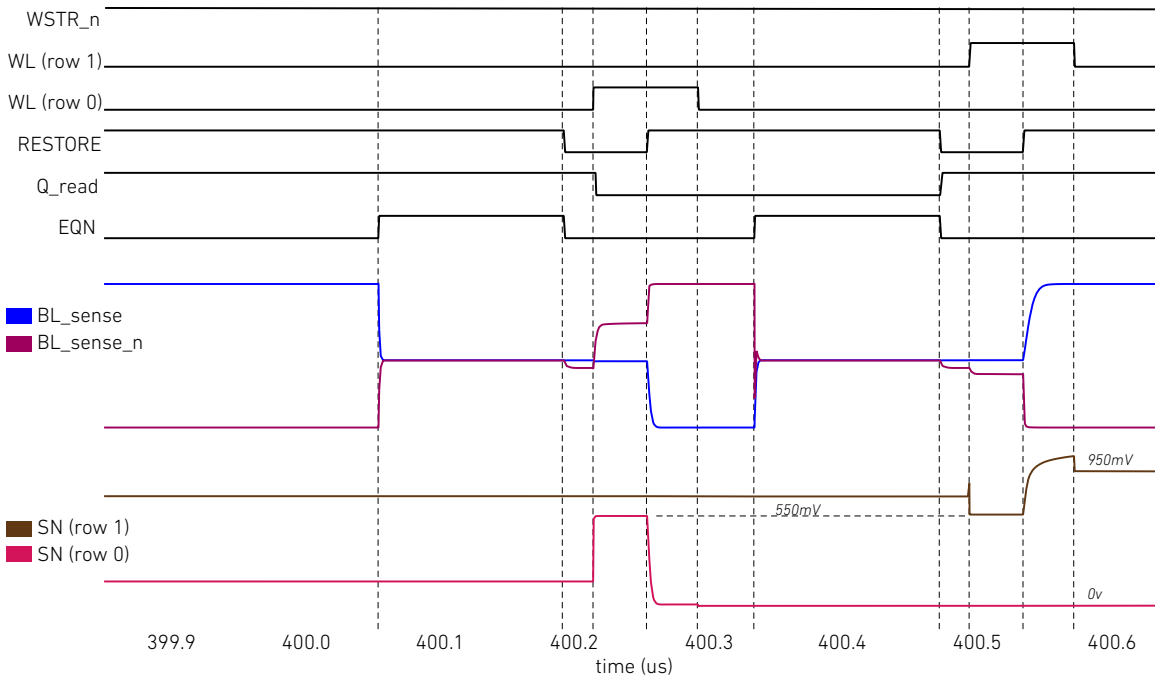


Figure 6.9 *Refresh simulation for two adjacent rows with different stored data.*

allowing the output of the second inverter to rapidly swing above the inverting threshold (This can be seen at the 400.2μs mark in the simulation).

- **Restore-1:** The *restore* switch is closed, thus allowing the sense amplifier to completely drive of the bit-line to logic value of the stored data.

- **Restore-2:** The *WL* is turned off, allowing the charge under the gate of *M2* to be trapped. Notice just before the 400.6*mu*s mark in the simulation, the closing of the *WL* causes charge injection in the storage node.

- **Idle:** The *restore* switch is left closed, therefore the last refreshed data is "buffered" in the sense amplifier and can be read-back at any moment.

Similarly, a timing diagram for the write operation is shown in Fig. 6.8b and will be described below:

- **Equalize-1:** Before starting a write/read sequence, the sense amplifiers are "equalized" by closing the *eqn* and *restore* switches.

- **Equalize-2:** After the bit-line has settled to the trip point of the cross-couple inverters, both switches are opened, leaving the bit-line un-driven, and floating at approximately $Vdd/2$.

- **Sense-1** Next, the *WL* is turned on and the *write_strobe* signal is pulsed high (and subsequently low), allowing the stored charge to partially charge the bit-line parasitic capacitance towards its logic value.

- **Sense-2** The sense amplifier promptly amplifies this perturbation, allowing the output of the second inverter to rapidly converge to the full-swing of the logic value of D_in.

- **Restore-1:** The *restore* switch is closed, thus allowing the sense amplifier to completely drive of the bit-line to D_in.

- **Restore-2:** The *WL* is turned off, allowing the charge under the gate of *M2* to be trapped.

- **Idle:** The *restore* switch is left closed, therefore the last written data is "buffered" in the sense amplifier and can be read-back at any moment.

Bias-Gated Operational Transconductance Amplifier (OTA)

The schematic for the OTA is illustrated in Fig. 6.10. The main feature of the circuit lies in the specialized bias-gating block which is used to disable the tail current in the OTA. When the *bias_en* signal is low, the gate of the bias transistor is routed to ground thus disabling the tail current. While the bias is disabled, the ML is pulled to Vp, which allows the CID array to operate normally; recall that for read/write/refresh operations, the math-line is required to be at Vp. Conversely, when the CID array is ready to compute, the OTA is enabled and allowed to settle. Stability was accomplished by introducing

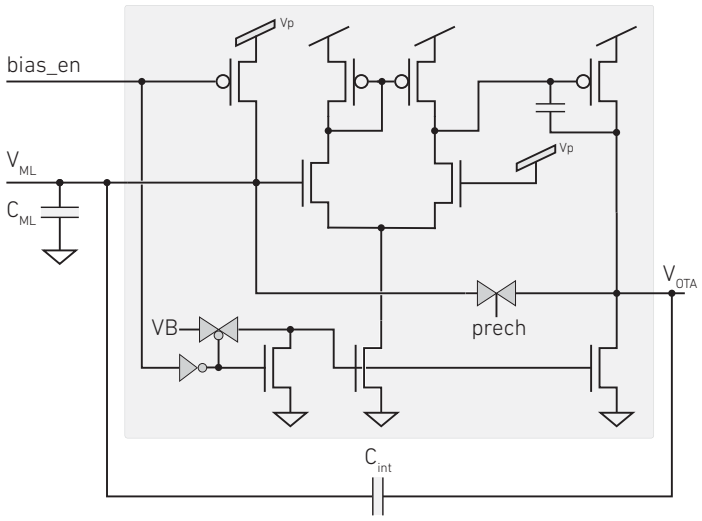


Figure 6.10 *Operational Transconductance Amplifier (OTA) used to pre-charge the math-line.*

a *Miller* capacitance in the second stage if the amplifier, also known as the *pole-splitting* technique.

DRAM-CID Compute

An equivalent circuit for a CID compute operation is shown in Fig. 6.11, where non-relevant circuitry was either grayed out, or removed for clarity. The sense amplifiers and access transistors do not play any part in the compute process. In turn, several switches are connected to the gate of the $M3$, providing appropriate input voltage swings, which are translated as a change of the internal charge of the circuit. Since the *prech* switch is open, the integration capacitor provides negative feedback, thereby clamping the math-line at $V_{ML} = Vp + V_{offset}$. Following this train of thought, when a positive voltage swing is presented at the gate of $M3$, any excess charge in the circuit will integrate in the capacitor, causing the output of the pre-charge operational transconductance amplifier (OTA) to swing towards ground. Provided that the OTA has converged (based on its slew rate and frequency response), an analog to digital conversion can take place. In this work, a single-slope A/D was utilized, and will be explained in detail in Section 6.3.3.

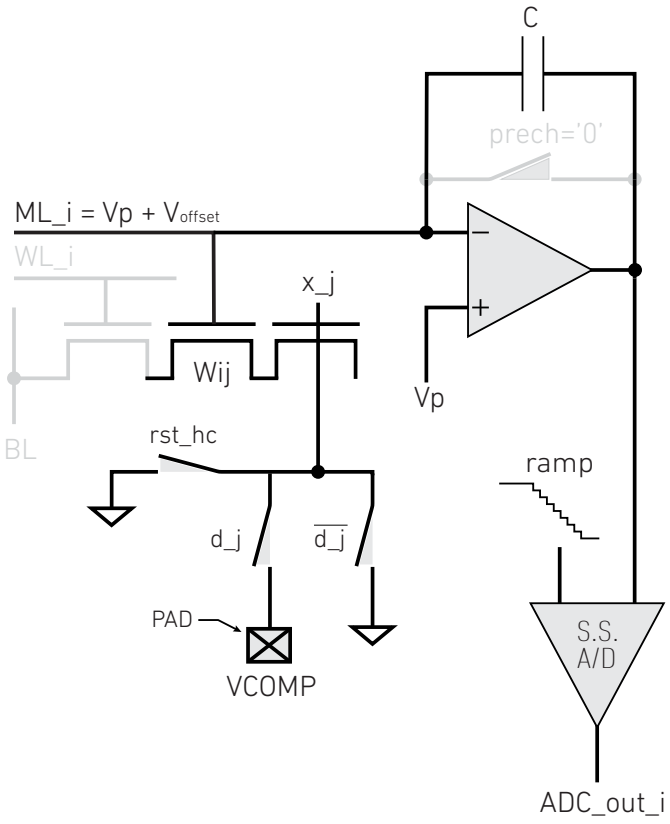


Figure 6.11 *Equivalent circuits for the CID compute operations.*

During a compute cycle, the input voltage swing may be either positive or negative, providing the capability to perform two quadrant multiplications. A polarity configuration bit is used on a column-basis, and by default is disabled.

Adiabatic Computing

A globally routed *rst_hc* switch provides an "override" to ground of the **x** inputs. This experimental configuration is shown in Fig. 6.12 and will be used to explore adiabatic recovered energy logic [84] [85]. From the figure, it can be seen that the *PAD* node is connected to *VCOMP* (typically $Vdd/2$ in this configuration) via an appropriately sized inductor (external). In this way, the circuit's internal nodes are reset with the *rst_hc* switch, causing the equivalent LC circuit to oscillate. The equivalent capacitance depends on the internal state of the d_j switches, therefore the frequency of the reset signal needs to be adjusted appropriately.

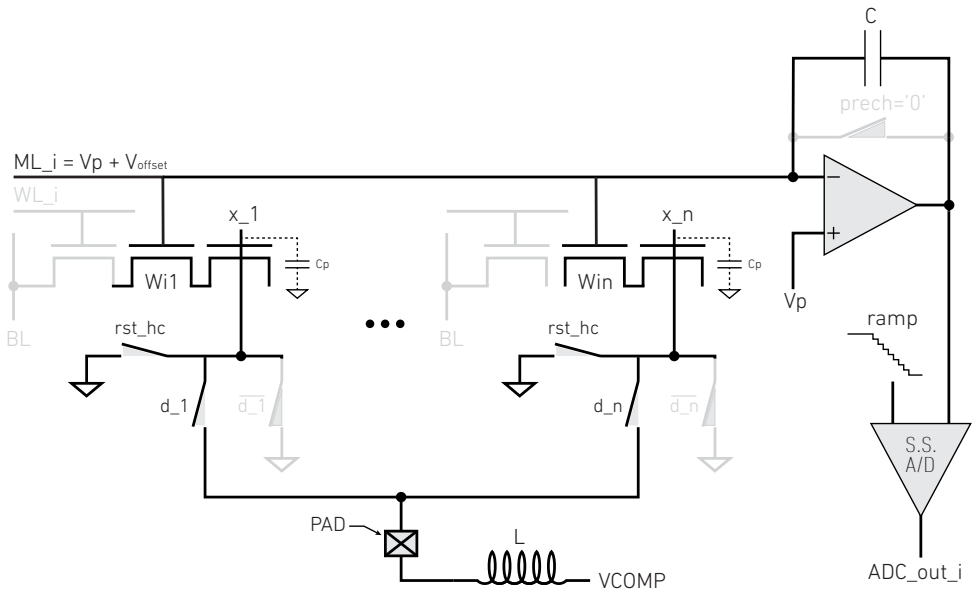


Figure 6.12 *Equivalent circuits for the CID compute operations using adiabatic recovered energy logic.*

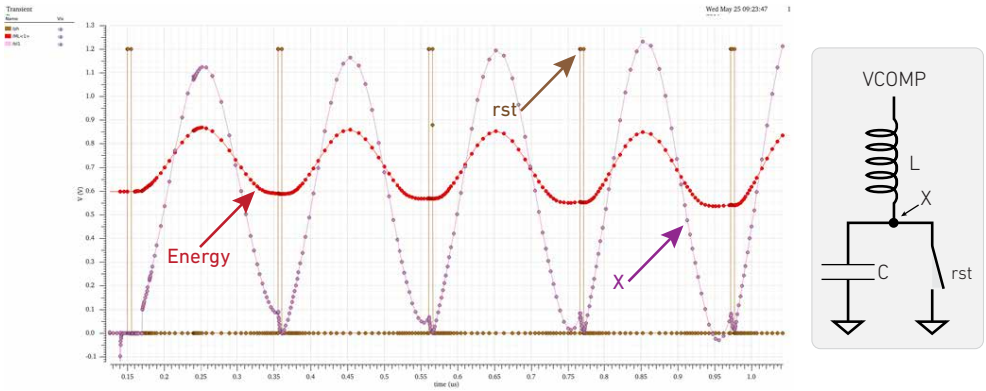


Figure 6.13 *CID array simulated while using adiabatic energy recovery.*

The *LC* oscillation provides adiabatic (slow with respect to the circuits' RC constant) charging, followed by a discharge and reversal in the current direction, thus recovering the charge.

Figure 6.13 shows a simulation of a 512×128 CID array while computing in adiabatic energy recovery mode. The circuit's energy was measured by integrating the instantaneous power of the circuit, clearly showing that charge flows to and from the inductor into the equivalent capacitance. The figure shows that the *rst* signal has a slightly different period than the oscillation frequency, thus causing

the internal **x** node to be pulled down out of phase.

6.3.3 A Single Slope Analog to Digital Converter (SSL ADC)

A row-parallel single-slope analog to digital converter (SSL ADC) was designed to convert the analog output of the pre-charge OTAs into meaningful digital values. Several design constraints were considered when choosing the A/D and are listed below:

- **Area:** The main driving factor for computational density is area. The repeated circuitry per row of a SSL ADC consists of a small differential comparator, amounting to not more than a few dozen transistors.
- **Power:** Power and area tightly coupled, especially when it comes circuits including capacitors, such those in successive approximation ADCs. In the current SSL design, no capacitors were used, and the number of biased stages was reduced to one, the comparator.
- **Leakage:** As shown in Chapter 4, leakage is present during CID computations, thus the A/D of choice needs to compensate for these effects. The SSL ADC designed in this work automatically compensates leakage. This is accomplished by creating the analog ramp with an additional CID row with its same electrical and parasitic characteristics.
- **Input range:** The valid range of the CID compute depends on two main factors: (i) the compute voltage, $VCOMP$, and (ii) the pre-charge voltage, Vp. The SSL ADC designed in this work naturally accommodates both factors. Since the analog ramp is created with a CID row, the initial value is the same as that in the computational crossbar (Vp). Moreover, the compute voltage ($VCOMP$) used in both analog ramp and CID is the same, thus the absolute valid range in both blocks will always be the same.

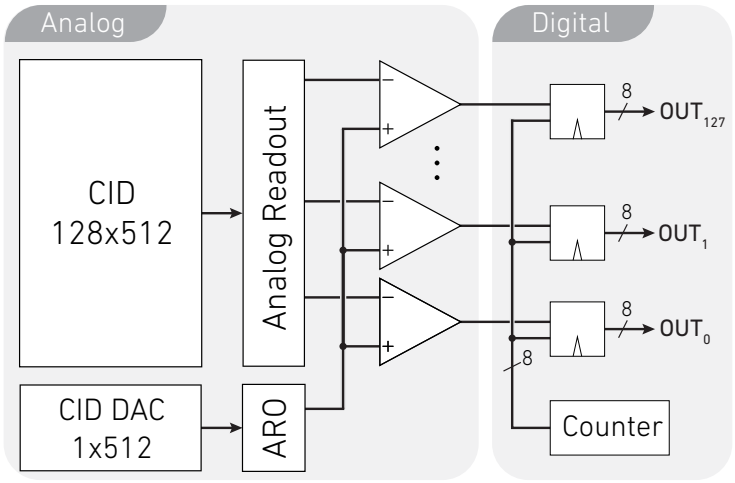


Figure 6.14 *CID array with row-parallel single-slope ADC (SSL).*

- **Non-uniformity correction:** The digital ramp in the SSL has been fitted with a piece-wise-linear function, which allows global offset and gain compensation.

SSL ADC: Principle of Operation

A high level block diagram of CID array and the SSL A/D converter is shown in Fig. 6.14. The left of the figure shows the analog circuits and the right hand side shows the digital periphery. Once the analog readout has been established, an analog ramp is broadcast across the array and compared with the compute values. The analog ramp is synchronized to a digital counter, whose value is also broadcast to a bank of registers. The counter value is latched in dedicated registers when the comparators trigger. A more detailed diagram of the SSL ADC with its internal circuits is shown in Fig. 6.15, and described below:

Analog Ramp: CID-DAC

The top-left block in Fig. 6.15 shows the internal circuits for the DAC (analog ramp). The core DAC consists of a dedicated CID-row built with 8 dummy rows on the top and 7 on the bottom. The DAC does not require refresh circuits; before an A/D conversion, all CID-cells are written with logic ones and are

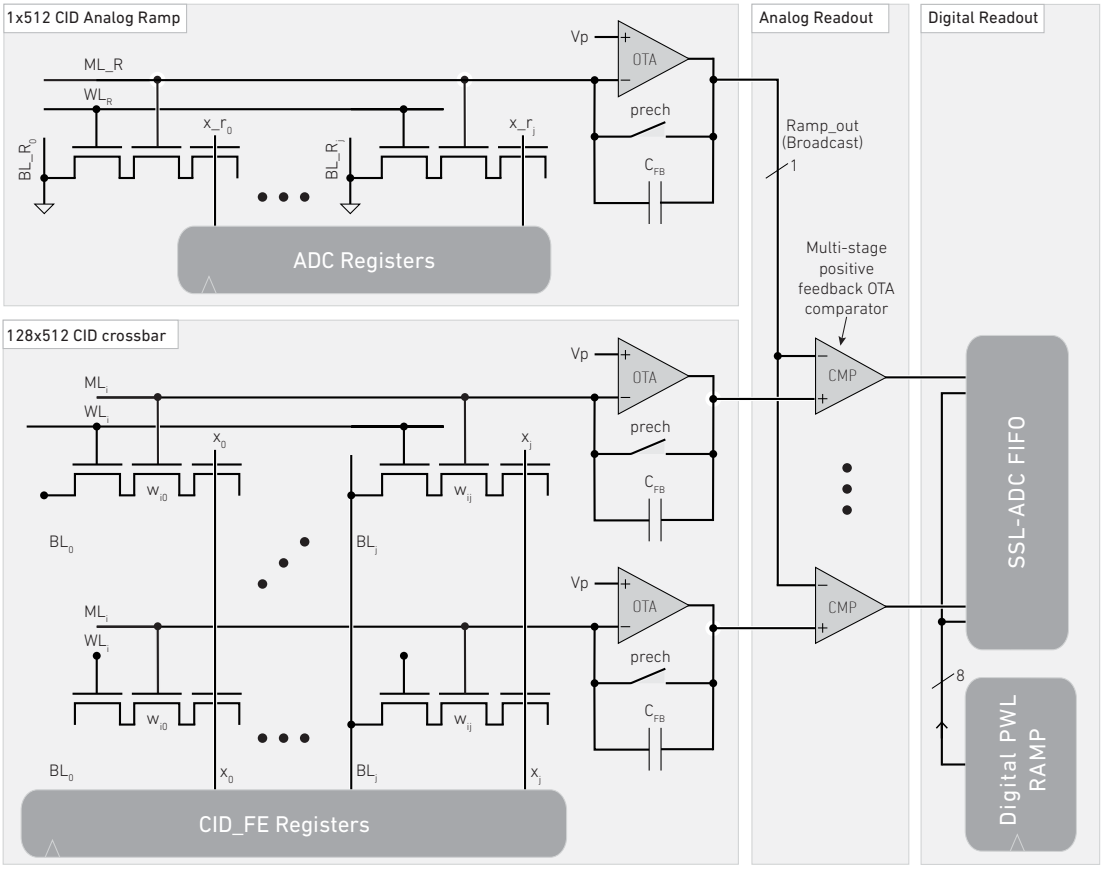


Figure 6.15 *CID array with internal circuits for row-parallel single-slope ADC (SSL).*

re-written until the compute cycle has competed ($\sim 2\mu s$ for 8-bits). The *ADC Registers* are controlled digitally, and provide arbitrary shifting (loading) of logic ones in the compute inputs, x_r_i, which in turn cause quanta of charge to integrate in C_{FB}. The DAC-ramp starts at the same voltage as the CID-compute and decreases to the maximum available excursion (minimum absolute voltage) given by $VCOMP$ and Vp. Digital control allows flexible pre-loading of the *ADC Registers*, thus providing offset compensation. Moreover, as data is shifted into the registers and *Ramp_out* decreases, leakage in the DAC-DRAM has proven to cancel the leakage in the Compute-DRAM.

Analog Readout: Comparator

The analog readout consists of a row-parallel bank of comparators. The goal of

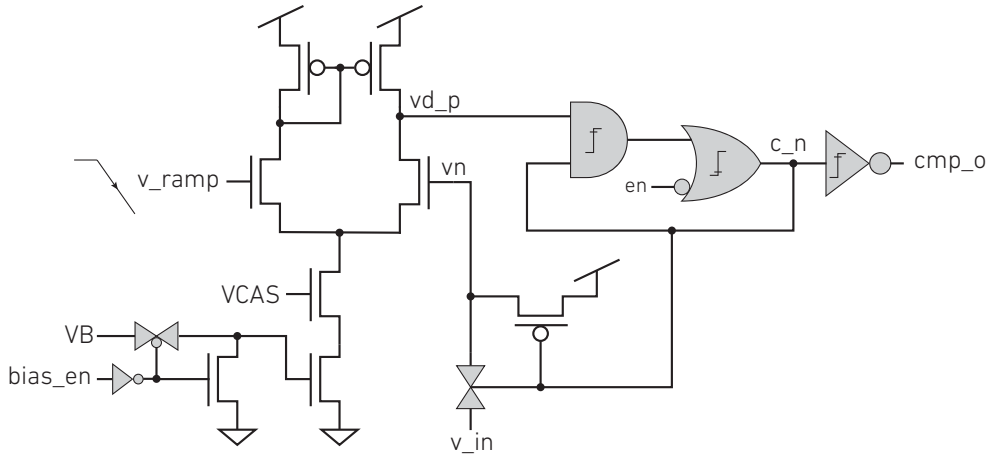


Figure 6.16 *Comparator used in the SSL ADC.*

the comparators is to provide a *write_enable* signal for loading the counter value in output FIFO. Power, area, voltage range, and response time was considered when designing the comparator, which resulted in the positive feedback multi-stage amplifier depicted in Fig. 6.16. The bottom left of the figure shows the bias-gating circuit, which allows the digital controller to shut-off the tail current of the differential pair while it is not in use. A cascode transistor in the bias path provides higher immunity to common mode interference and mismatch. The *enable* signal is used to gate the output to avoid glitches during setup (ramp reset and bias enabling/disabling) and is set high prior to a comparison. The circuit operates under the assumption that the non-inverting input, v_ramp starts high, and decreases monotonically. The non-inverting output, vd_p, will remain at Vdd, therefore node c_n will remain high. The former causes the AND gate and the v_in transmission gate to remain on. When v_ramp approaches v_in, vd_p reaches the threshold of the AND gate (which is set to be close to vdd via sizing), causing it to start shutting off. This effect propagates through the OR gate and causes the inverting input, vn, to pull up to Vdd. This positive feedback causes the non-inverting output of the differential pair to pull-down even faster, whose effect propagated to the output. The chain of gates was

designed with logic thresholds in a high-low-high sequence, creating higher gain, and speed.

The comparator was designed to converge in less than 5ns, verified for the full input range (1v down to 0.5v) via Montecarlo simulation. Additionally, only a 200nA bias current is required during the comparison time, which makes the comparator burn 240nW of static power, yielding 0.48pJ per 8-bit conversion.

Compute Operation

Figure 6.17 shows a timing diagram of a compute operation. The figure shows the analog ramp (V_ramp) in red, and the CID compute output of the M^{th} row (V_CID) in blue. The *ADC Registers* (x_r) are sequentially loaded with logic ones in spatial increments (unary) of Δr. A digital ramp is synchronized to the loading sequence of the *ADC Registers*, keeping track of the analog voltage change. When the comparator trips, the contents of digital counter is latched in the *ADC* register. Both analog and digital ramp are broadcast to the entire array, therefore performing up to 128-fold 8-bit conversions in parallel.

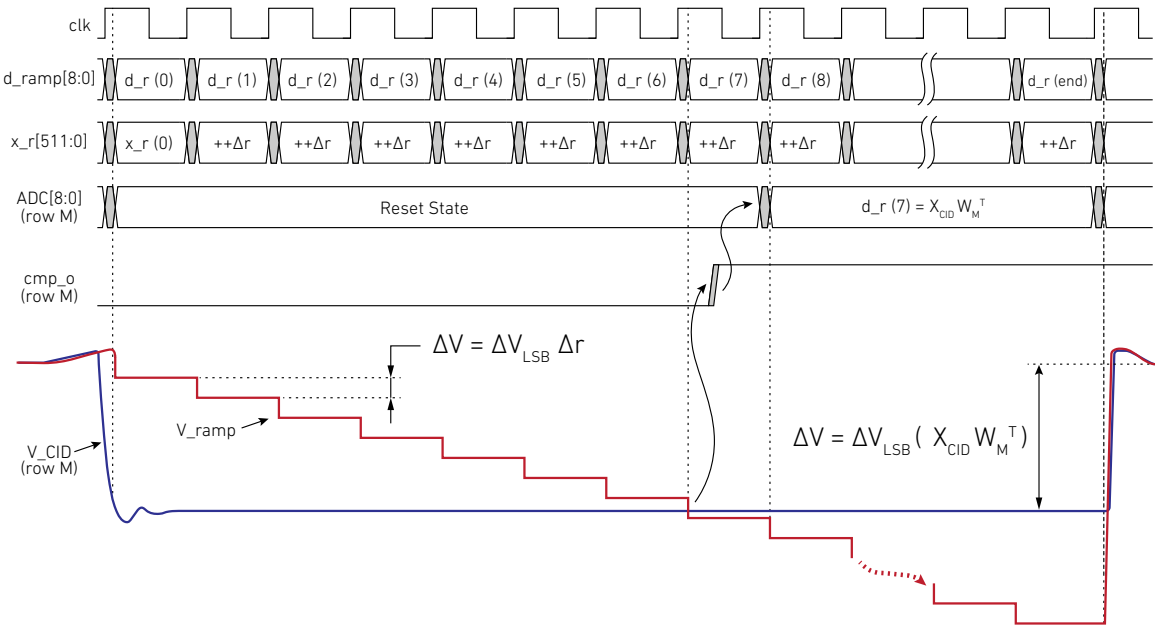
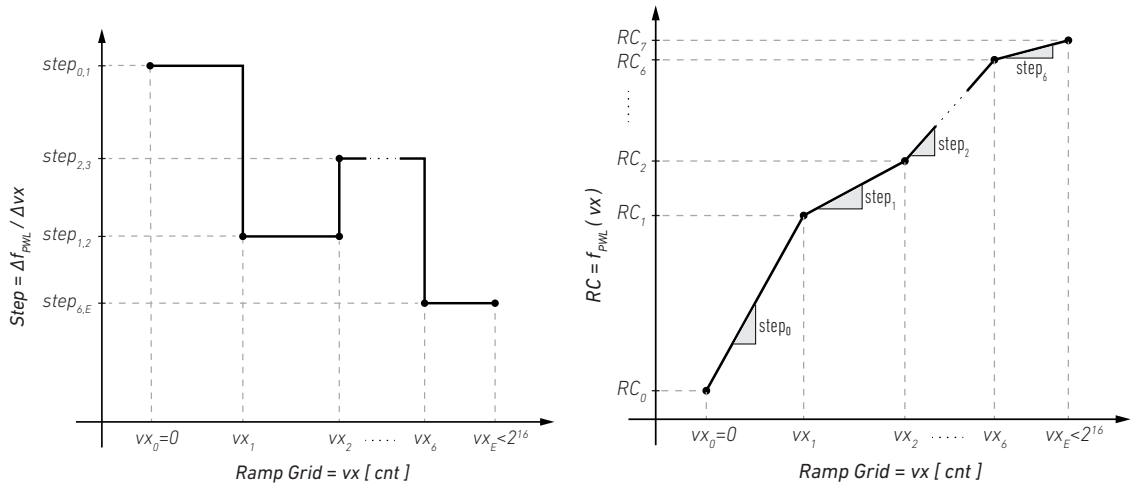


Figure 6.17 *Timing diagram for ADC conversion.*



(a) (b)

Figure 6.18 ***(a)*** *The x-axis represents the* domain *of the Piece-Wise-Linear (PWL) function, and consists a set of vertexes representing the time at which the* Step *(or increment) of the PWL function changes. The y-axis represents the first derivative of the PWL function, f'_{PWL}. The units of the x-axis are given in* counts*, which represent the state of a digital counter which increments by 1 at a divided frequency of the system clock.* ***(b)*** *This graph shows the state of the Ramp Counter ($RC = f_{PWL}$) as a function of the* Step *value and its vertexes (vx).*

Digital Piece-Wise-Linear (PWL) Ramp

The digital ramp was designed to have 8 fully programmable slopes in addition to an offset setting. This function provides a global non-uniformity correction mechanism allowing to compensate for non-linearities in the OTA readout and systematic offset in the comparators. The vertexes for the PWL function are shown in shown in Fig.6.18a, whose values are programmed via the AHB bus. The resulting PWL ramp is shown in Fig. 6.18b, where the different slopes are applied at arbitrary time-locations (in clock cycles). All parameters pertinent to the SSL-ADC can be found in Tables F.14 & F.15.

On a final note, a Successive Approximation Register (SAR) ADC was designed and connected to the last row of the CID array and will allow to cross-check the behavior of the array against two different converters. The design of the SAR-ADC was carried out by Christos Sapsanis at the Johns Hopkins University, and escapes

the scope of this thesis.

6.3.4 A Custom Processor for Controlling the DRAM and Computational Memory

Mixed signal blocks such as DRAMs are challenging to control, thus post-silicon validation has shown that signal timing requirements often change when compared to simulation. In test-chips, critical control signals are usually wired to the periphery, thus allowing external controllers to be tailored to unforeseen effects in internal circuits. Since the ESC Core is part of a larger array processor, off-chip signaling was not an option, hence, a flexible controller was designed considering power and area constraints.

A custom 1-stage pipelined instruction set processor was designed to perform all control and compute related tasks in the CID array: the Pico-Controller. A block diagram of the Pico-Controller is shown in Fig. 6.19 and comprises the following main blocks:

- **instruction memory (IR),** which is addressed by the program counter, and bootstrapped by the AMBA bus.

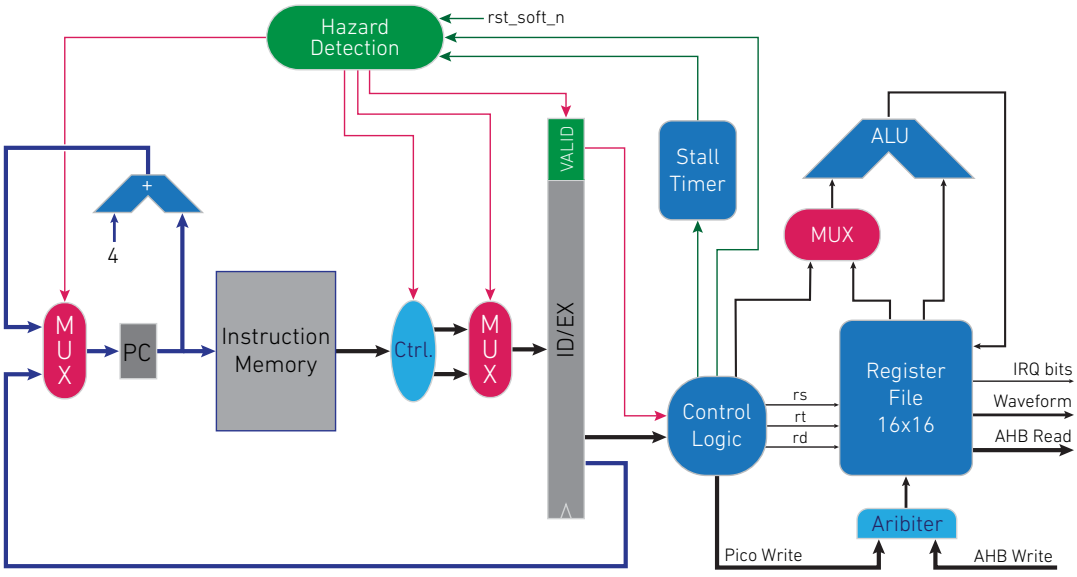


Figure 6.19 *Pico-Controller architecture.*

- **Control unit:** which is in charge of decoding instructions and controlling data flow.
- **Register file:** which has dedicated and general purpose registers.
- **ALU:** In charge of calculating jump addresses and performing any on-demand arithmetic tasks.
- **Stall timer:** which is a real-time counter used to create pipeline stalls for instruction decoding.
- **Hazard unit:** which is in charge of flushing the pipeline or holding valid instructions when appropriate.

The Pico-Controller is responsible for generating signal patterns that control the read-/write/refresh and compute operations, as well as providing IRQs to the MPU/DMA and control signals for several analog blocks. A bus arbiter between the Pico-Controller and the *AHB Subsystem* allows any bus master to write to the instruction memory, mainly used for bootloading.

Instruction Set Architecture (ISA)

Table 6.1 shows the Instruction Set Architecture (ISA), which was designed to support single cycle granularity for copying data from the instruction memory to the register file. Table 6.2 shows the corresponding assembler, which encompasses 9 instructions in 4 distinct formats. A dedicated register, *Waveform Register*, is wired directly to the control signals of the mixed signal block, thus allowing direct control over the analog portion of the system. The ISA supports loops and stalls, providing arbitrary timing between 1 and 256 clock cycles on the *Waveform Register*, and additionally has some arithmetic and logic functions, which can be utilized as needed.

Table 6.1 *Instruction Set Architecture (ISA) of the Pico-Controller. The Pico-Controller supports 4 instruction formats (**I2R, R2R, JUMP, STALL**) and a total of 9 operations. Additionally, the processor has a special **STALL** instruction which is used to program an internal counter which is not part of the ALU.*

Format	**Opcode**	**Instruction Bit**																									
		25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I2R	ADDI	0	0	s				t				imm															
R2R	ADD	0	1	s				t				d				×	×	×	×	×	×	×	×	×	0	0	0
	SUB	0	1	s				t				d				×	×	×	×	×	×	×	×	×	0	0	1
	OR	0	1	s				t				d				×	×	×	×	×	×	×	×	×	0	1	0
	NOR	0	1	s				t				d				×	×	×	×	×	×	×	×	×	0	1	1
	XOR	0	1	s				t				d				×	×	×	×	×	×	×	×	×	1	0	0
	AND	0	1	s				t				d				×	×	×	×	×	×	×	×	×	1	0	1
JUMP	JEQ	1	0	s				t				imm															
STALL	STALL	1	1	×	×	×	×	×	×	×	×	imm															

Table 6.2 *Assembler and operation of the Pico-Controller.*

Format	**Opcode**	**Assembler**	**Operation**
I2R	ADDI	addi rt, rs, imm	$rt = rs + imm$
R2R	ADD	add rd, rs, rt	$rd = rs + rt$
	SUB	sub rd, rs, rt	$rd = rs - rt$
	OR	or rd, rs, rt	$rd = rs \mid rt$
	NOR	nor rd, rs, rt	$rd = \sim (rs \mid rt)$
	XOR	xor rd, rs, rt	$rd = rs \oplus rt$
	AND	and rd, rs, rt	$rd = rs \cdot rt$
JUMP	JEQ	jeq rs, rt	jump *if* $(rs = rt)$
STALL	STALL	stall imm	stall pc #imm

Memory Map

Table F.4 in Appendix F.2 shows memory mapped registers that are used in the Pico-Controller configuration. This peripheral lies in the ESC Core and ESC Subsystem module, respectively. The register file can be accessed from external agents, thus allowing appropriate data exchange enabling fine control of subroutines and interrupt interaction. Registers and bits mapped in the Pico-Controller memory space are described in Table F.5 in Appendix F.2.

Register File

Table F.6 in Appendix F.2 shows all 16 registers that belong to the register file of the Pico-Controller. Each register is a half-word (16 bits), although the address space accounts for 32-bit words. All registers can be read and written from both Pico-Controller and MPU/DMA masters. The *Row Addr.* and *Column Addr.* Registers are of particular importance; thus, they are used by both Pico-Controller and MPU/DMA to control the column and row addresses of the CID array.

Waveform Register

Table F.7 in Appendix F.2 shows all bits of the *Waveform Register* with their respective descriptions. Bits in the *Waveform Register* are used to control analog portions of the CID. Since many of the analog blocks are sensitive to timing, this register is used as a fine-grained control mechanism with single clock-cycle granularity and is the main reason for the existence of the Pico-Controller. Waveform templates for writing, reading and computing are described in Table F.8.

Status Register

Table F.9 in Appendix F.2 shows all 16 bits the *Status Register* along with their description. The *Status Register* is used as a means for the Pico-Controller to interact with the MPU/DMA as well as the ADCs in the core. In particular, the interaction between the DMA and the status register is described in Table F.10.

6.3.5 ESC Subsystem Digital Peripherals

Core Memory Map

Table F.11 describes the ESC Core memory mapped modules. The main component in the ESC Core is the CID crossbar, which is controlled by a custom instruction set processor: the Pico-Controller. In this context, the Pico-Controller's instruction

memory is mapped as a peripheral on the AMBA Bus as well as its configuration registers. This allows an external agent to bootload and configure the Pico-Controller in a hassle-free way. Noteworthy, the AMBA bus may only write to the instruction memory if the Pico-Controller is in reset state.

CID Front End Registers

The CID Front End (CID_FE) peripheral consists of a group of memory mapped registers that allow the digital system to interface with the analog portion of the ESC Core. On the one hand, data may be copied from main memory (TIM) to the front end registers, which in turn are used to control the data input to the refresh circuits, which allow writing parameters and computing in CID array. On the other hand, after a read operation, data may be copied back to main memory using the same address space. The memory mapped registers CID Front End are shown in Table F.12 and their respective description is shown in Table F.13 in Appendix F.3.

Control (SSL ADC)

A custom controller dedicated to the generation of the analog and digital ramps was designed as part of the ESC Subsystem. This block interacts with the Pico-Controller, DAC registers, and the output ADC FIFO. Table F.14 shows all memory mapped registers in this peripheral and Table F.15 gives a detailed description of each signal.

FIFO (SSL-ADC)

The analog to digital converter explained in this chapter requires a 1-stage FIFO peripheral which is mapped in memory and allows the bus master to fetch computation results. This block interacts directly with the Pico-Controller, which in turn communicates with the bus master via IRQ to indicate the completion of a compute cycle. Even though the FIFO has a depth of one, it communicates with its periphery in a standard way, using full and empty flags. The full flag is global to the peripheral and

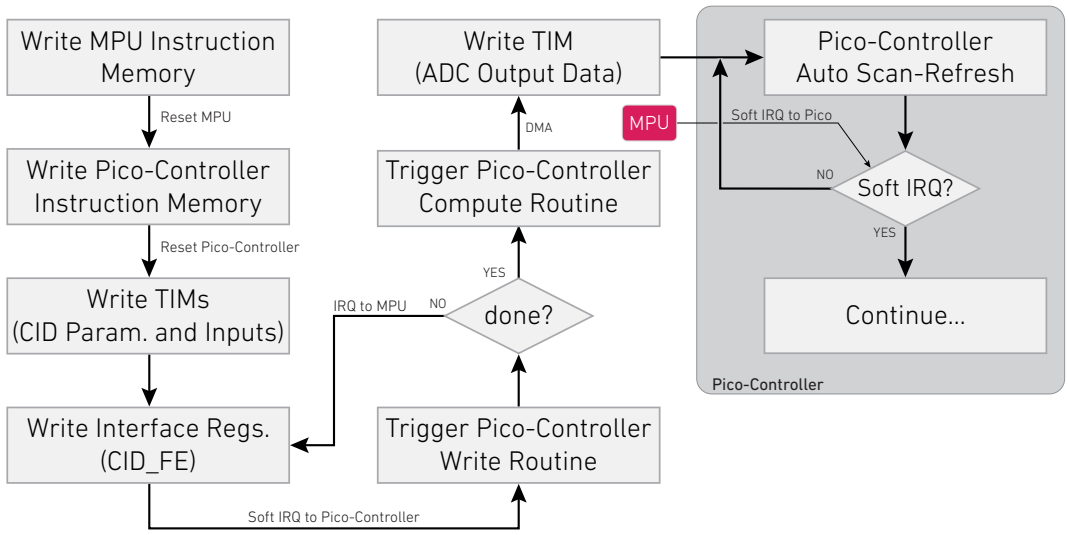
is the result of the reduction of all full flags ($\bigcup_i \text{full}_i$) and is both mapped in memory and wired to the periphery as an independent signal so it can be used as an IRQ.

6.3.6 ESC Subsystem Data Flow

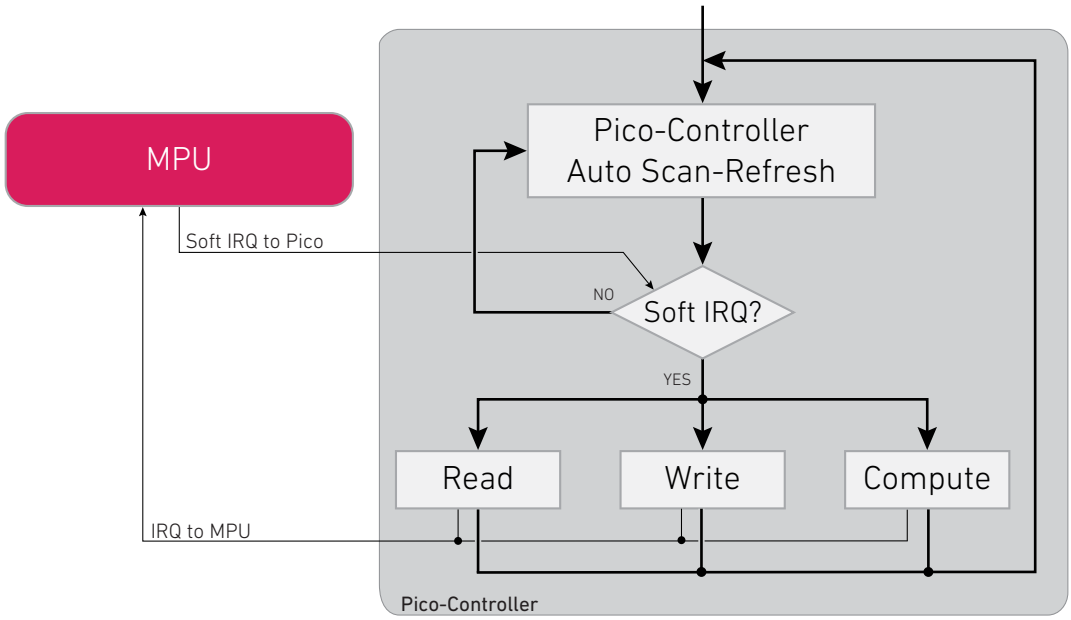
This section addresses the data flow required to operate a CID array. Although programming procedures for the NoC and MPU are out of the scope of this thesis, they will be explained on a high level, allowing the reader to comprehend the required steps to utilize the ESC Subsystem within the context of the SoC (processor array).

Firstly, let us consider an empty CID array; data must be copied to all the peripherals listed in Section 6.3 (*Pico-Controller, CID_FE Registers, SSL-Control, etc*) from either local memory (TIMs), or external memory (NoC). The memory transfers are carried out by either the MPU or the DMA unit. Once the Pico-Controller has been booted, it can be used to generate the necessary waveforms to perform a write operation, and subsequently report back to the MPU/DMA via interrupt; this process is repeated for as many rows as desired.

At this point, the array may be used to compute a VMM or merely to recall data (albeit with high read latency). On a row-by-row basis, the crossbar is capable of computing a 128-fold 512-length 1-bit normalized dot product between an input and stored data. The computation result is exhibited on an integration node (Math-Line) and subsequently converted to digital domain using a row-parallel bank of single slope ADCs for post-processing. Figure 6.20a illustrates a general write/refresh routine that is used when storing parameters in the CID-DRAM array, and Fig. 6.20b shows a secondary data flow scheme where the MPU requests a read, write, or compute routine to the ESC Subsystem.



(a) *Write and auto-scan refresh.*



(b) *On-demand read/write/compute.*

Figure 6.20 ***(a)*** *Data flow for writing data to DRAM and executing an auto-scan refresh routine. The main idea is that the MPU and Pico-Controller runs a pre-loaded routine for auto-scan refresh until the MPU sends a* Soft IRQ. ***(b)*** *When the Pico-Controller receives a* Soft IRQ*, it jumps to the appropriate section and executes the on-demand routine (e.g. read contents of CID row 144). Upon completion, the Pico-Controller sends an IRQ back to the MPU and continues with its program.*

6.4 Results

6.4.1 Implementation

The SoC described in this chapter resulted in a 12×12mm layout, taped-out on a dedicated wafer in a CMOS 65nm logic process. A top-level layout of the SoC and the Core-Unit (CU) is shown in Fig. 6.21 and a micrograph of the bonded chip is shown in Fig. 6.22. At the moment of writing this thesis, the aforementioned bonded chip is being prepared for testing. The SoC (left) shows its internal blocks, which consist of two SiFive processors, 6 auxiliary units, a GPIO interface, a high-speed-interface (HSI), and 49 Core-Units. The Core-Unit (right) consists of two ESC Cores, which in turn contain the mixed-signal CID arrays. The Core-Unit was synthesized on a flat hierarchy, therefore there is no distinct line between the two bus-slaves.

Moving down in hierarchy, the CiM Core layout is shown in Fig. 6.23 and its top-level interface and signal description is described in Table F.16; the design was laid-out entirely by hand, amounting to 278K transistors. An area breakdown of the CiM core can be seen in Table 6.3; most of the area is occupied by the CID array itself, occupying nearly 68% of the core. The layout's pin-out is on three edges, allowing the

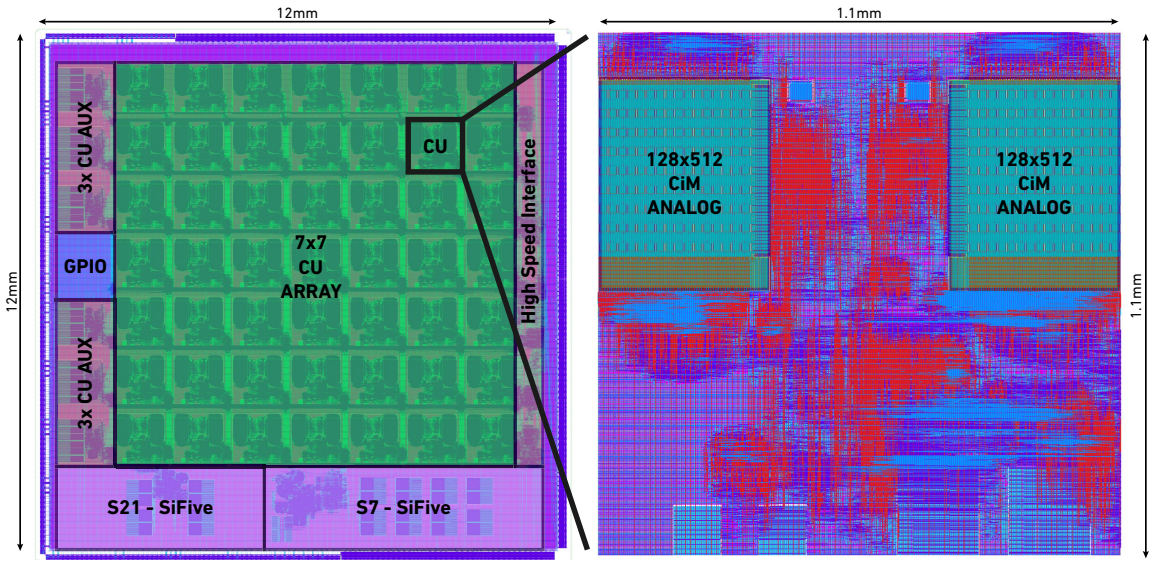


Figure 6.21 *System-on-Chip and ESC Subsystem layouts.*

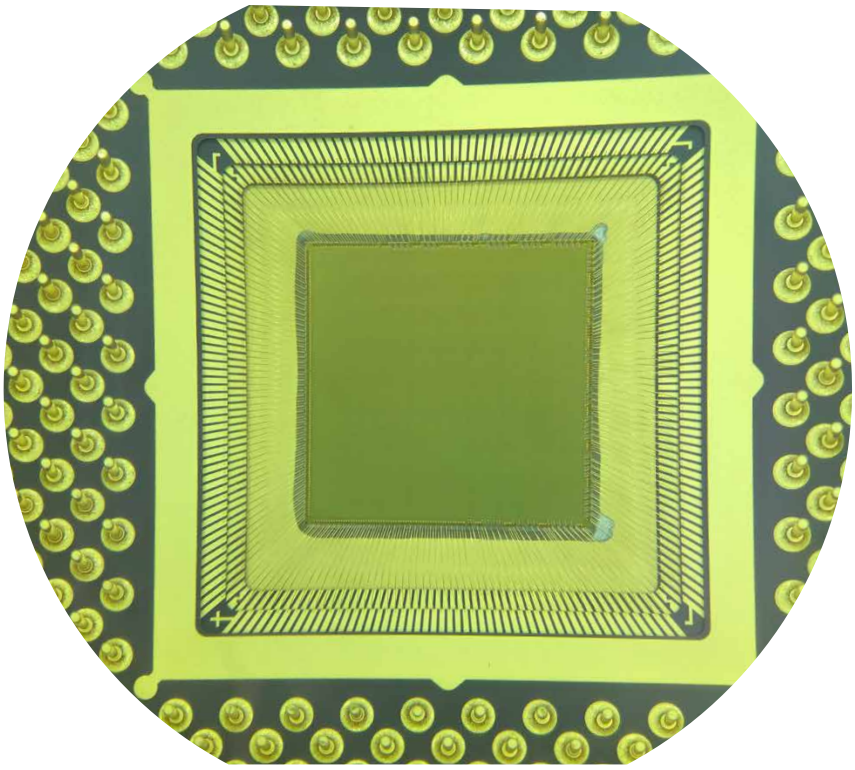


Figure 6.22 *System-on-Chip micrograph and showing bonded die in its package.*

Table 6.3 *Area breakdown of the CiM core.*

Block	**Area** μm^2	**Area %**
CID Array	101760	67.9
CID Control	22260	14.8
DAC	12720	8.5
SSL Comparator	7458	5.0
DAC Control	3000	2.0
CID Analog Readout	2740	1.8
Total	149938	100.0

cell to be abutted on the edge of the CU. The bottom edge provides connection to the sense-amplifiers and column control switches, the right edge provides row-control and readout, and the top edge provides access to the DAC array registers.

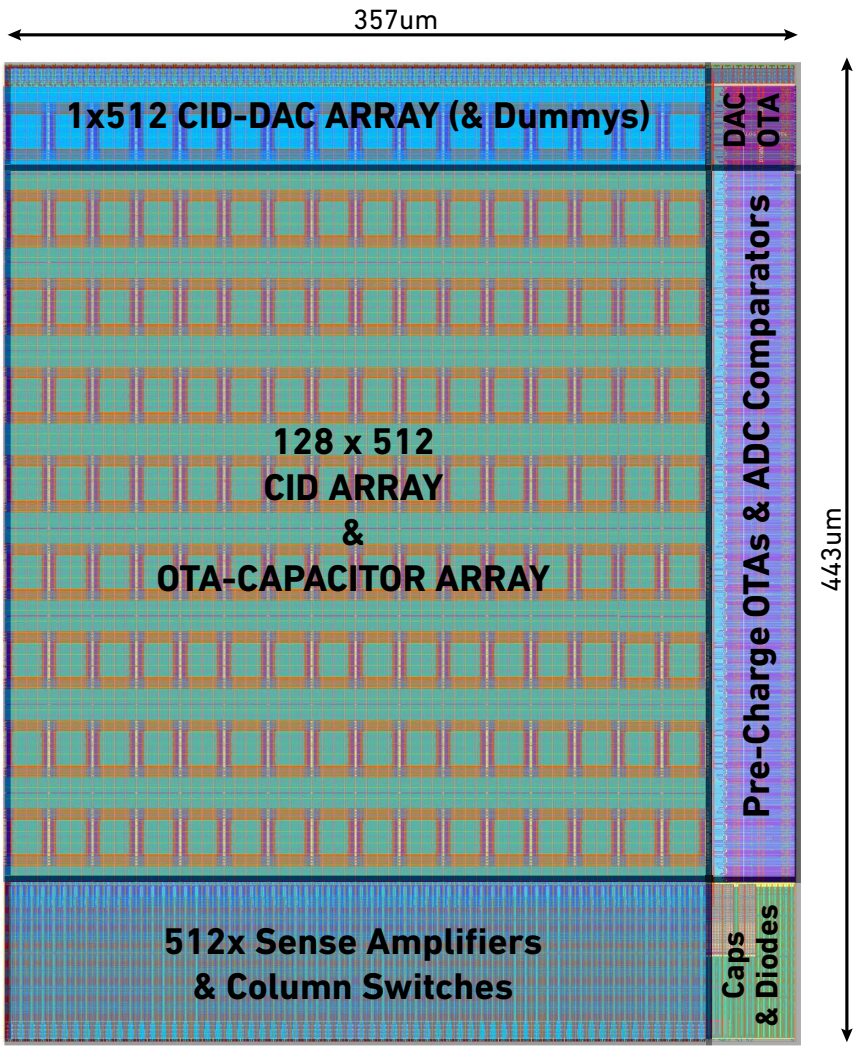


Figure 6.23 *Compute-in-Memory Core layout*

Design Considerations

The layout in Fig. 6.23 shows a 128×512 CID array (middle), where OTA capacitors (C_{FB}) were drawn in upper layers. Capacitors were routed in two metals (5,6) as vertical-natural alternate-polarity interdigitated devices, with individual isolation rings, and a ground plane. The ground plane provides isolation to the CID array, thus avoiding unwanted signal coupling. Each individual capacitor was routed to the OTA with same-length wires, providing identical capacitance loading.

Similarly, the sense amplifiers' sensitive signals were isolated in a co-axial fashion

and proven to be immune to parasitic coupling via Montecarlo simulation. Eight sense-amplifiers and control switches were bundled and pitch-matched to the CID columns, yielding a horizontal effective pitch of 0.62μm, which proved to be a routing challenge in 5 metals.

The top portion of the layout shows the CID-DAC array, which consists of 16 rows, out of which only one is used for the DAC. A total of 16 rows were included, providing adequate dummies for matching purposes in both CID cells and OTA-integration capacitor; all dummy circuitry was grounded. The DAC OTA's output was routed down vertically to the ADC comparators and shielded appropriately.

All differential amplifiers were designed to have balanced antennas; the ratio of polysilicon, metals and diffusion was set to satisfy recommended design rules and made equal for matched-pairs. This effect allows gate degradation to be uniform across select transistors during the ion-implantation step in the fabrication process. Additionally, differential pairs were matched using traditional analog techniques (sizing, common centroid layout, adding dummies, etc.).

The bottom-right of the layout shows a dedicated area were decoupling capacitors were introduced to supply current surges in the sense amplifier power rails. Since external biases are routed directly to gates, local tie-down diodes were introduced to satisfy chip-level antenna rules. In addition to adding tie-downs, bridge-routing techniques were used to decouple the antennas between the internal and external circuits.

6.4.2 Simulations

Sense Amplifier

The most critical circuit in the DRAM operation is the sense-amplifier, which is used in read/refresh and write operations. Figure 6.24 shows a *process + mismatch* Montecarlo simulation of the full DRAM array, where two rows with different data

were written, and iteratively accessed. The plot on the left shows the percentage of successfully refreshed logic zeros out of 1000 trials, amounting to 96%. Similarly, the plot on the right shows the percentage of successfully refreshed logic ones for the same amount of trials, amounting to 89%. The bottom plots show the respective transient responses of the storage nodes, where leakage appears as an exponential decay towards $Vdd/2$. The *process* option in the Montecarlo simulation ensures that the variability across wafers allows for a retention time of 400μs, while the *mismatch* option accounts for local parameter variation within a single wafer.

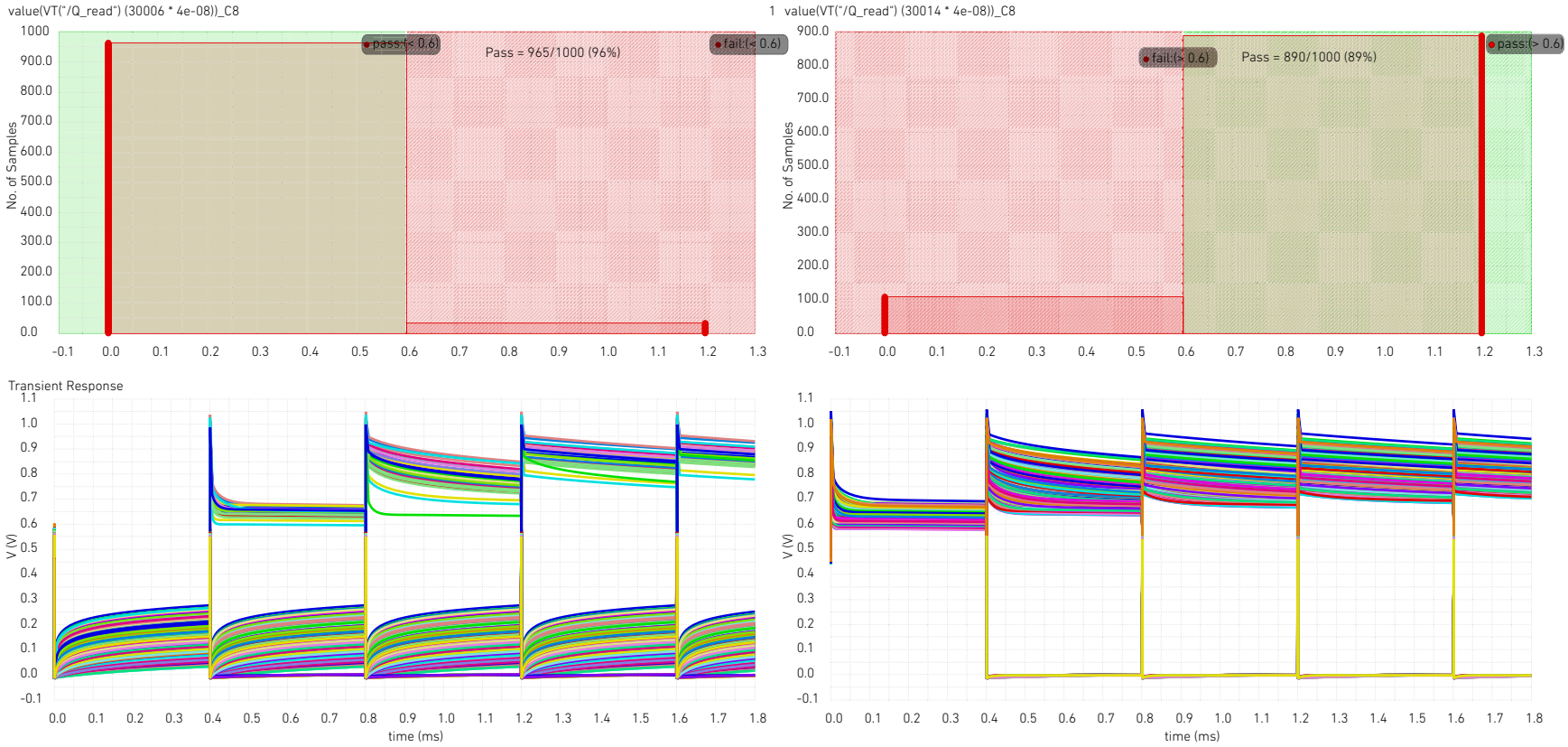


Figure 6.24 *Montecarlo simulation of refresh circuits.*

Compute Operation

Figure 6.25 shows a simulation of a single CID row. The full simulation shows a continuous iteration of *load*-**x** and *compute*, where the input values are increased linearly in steps of 16. The analog ramp was set to increment in steps of $\Delta r = 2$, thus creating an 8-bit DAC. The effective number of bits (ENOB) is hard to predict due to the high level of programmability of the controller (non-uniformity correction and PWL digital ramp) and is expected to reach 8-bits based preliminary simulation data.

From the figure, the red waveform shows the analog ramp over all iterations and the blue waveform shows the compute value, which is the output of the pre-charge OTA. The most important feature of the compute-an-convert sequence is that launched from the same resting voltage as the compute, and gradually leaks at the same rate as CID. The effect is clear in the blue waveform, where after a negative ΔV, the voltage increases linearly (even exceeding the resting potential in some cases).

When the ramp crosses the leaking compute output, the comparator triggers a one-shot circuit that in turn is used to latches the digital-ramp's (PWL) state in output FIFO (denoted *adc_out* in the simulation). For each set of increments in the analog-computation, the ADC outputs increment by 16, which is the same amount by which the input was iteratively increased. From the last compute cycle on the far right, it can be seen that after the ramp has reached its minimum value (maximum span), the full-scale compute reaches exactly the same voltage.

The simulation in the figure was carried out in the Analog-Mixed-Signal (AMS) environment, in Cadence ADE-XL, and required approximately 12 hours to complete running on a 24-core (48-thread) AMD Threadripper.

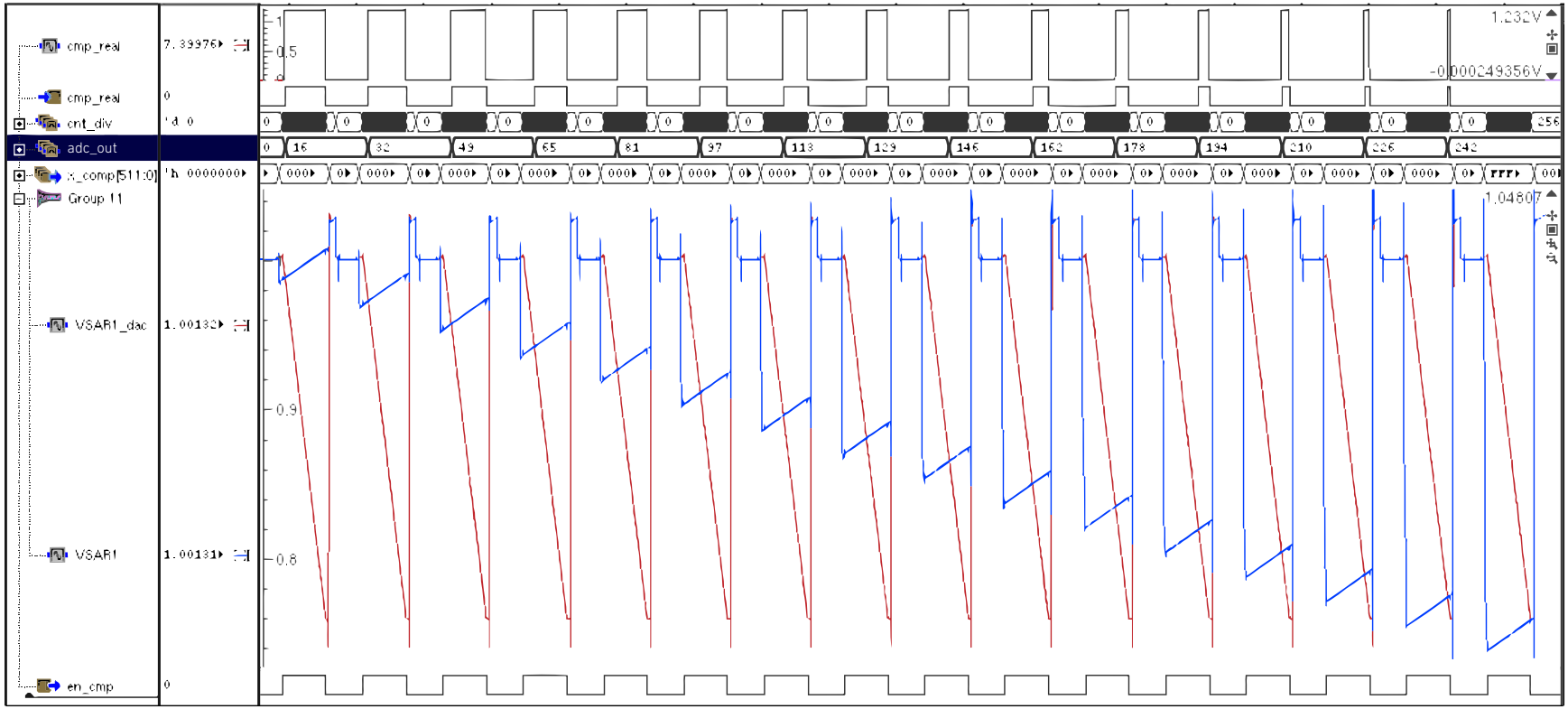


Figure 6.25 *Compute simulation of a single row for different input values.*

Full Post-Layout Compute-and-Refresh Array Simulation

Post-layout simulation is carried out by creating a hierarchical parasitic extraction (PEX) of the top-level analog block, including, power distribution, decoupling capacitors, and diodes. Parasitic coupling effects at the system level were used to iteratively optimize the entire design. It was found that running simulation based on real processing data (as opposed to idealized scenarios) revealed several transition cases that caused loss of data in the DRAM refresh procedure. Many nets sensitive to parasitic coupling were identified; ground planes and guard-rings were added to OTA capacitors and several co-axial shielding was added to sensitive wires, mainly in the refresh circuits and the broadcast of the analog ramp.

Figure 6.26 shows a post-layout (after parasitic extraction) simulation of an entire CID array (128×512) when incremental weights ($\mathbf{w_i}$) were stored in different rows and an all-ones input ($\mathbf{x}$) was applied to the array followed by a compute sequence. The figure shows the family of compute-outputs, which achieves 128-levels distinct levels. The curve in blue shows the analog-ramp, which suffers some non-linearity due to a the large parasitic load of the ramp OTA. This is revealed clearly at the launch of the ramp, where there exists a transient response before stabilizing.

From the simulation, it can be seen that the ramp does not reach its minimum value due to an interruption in the compute cycle; as a result, several converted outputs are inaccurate (in this particular simulation). Nonetheless, the ramp launch-time and compute-hold time are parametrizable, therefore the ramp can reach the required minimum with no issues. After the compute sequence was complete, a refresh scan was performed on the entire array, resulting in correct refreshed values 100% of the time.

The complete simulation took 3 weeks running on all 24-cores of an AMD Threadripper with 128GB of RAM. Analysis of the tools and simulation performance led

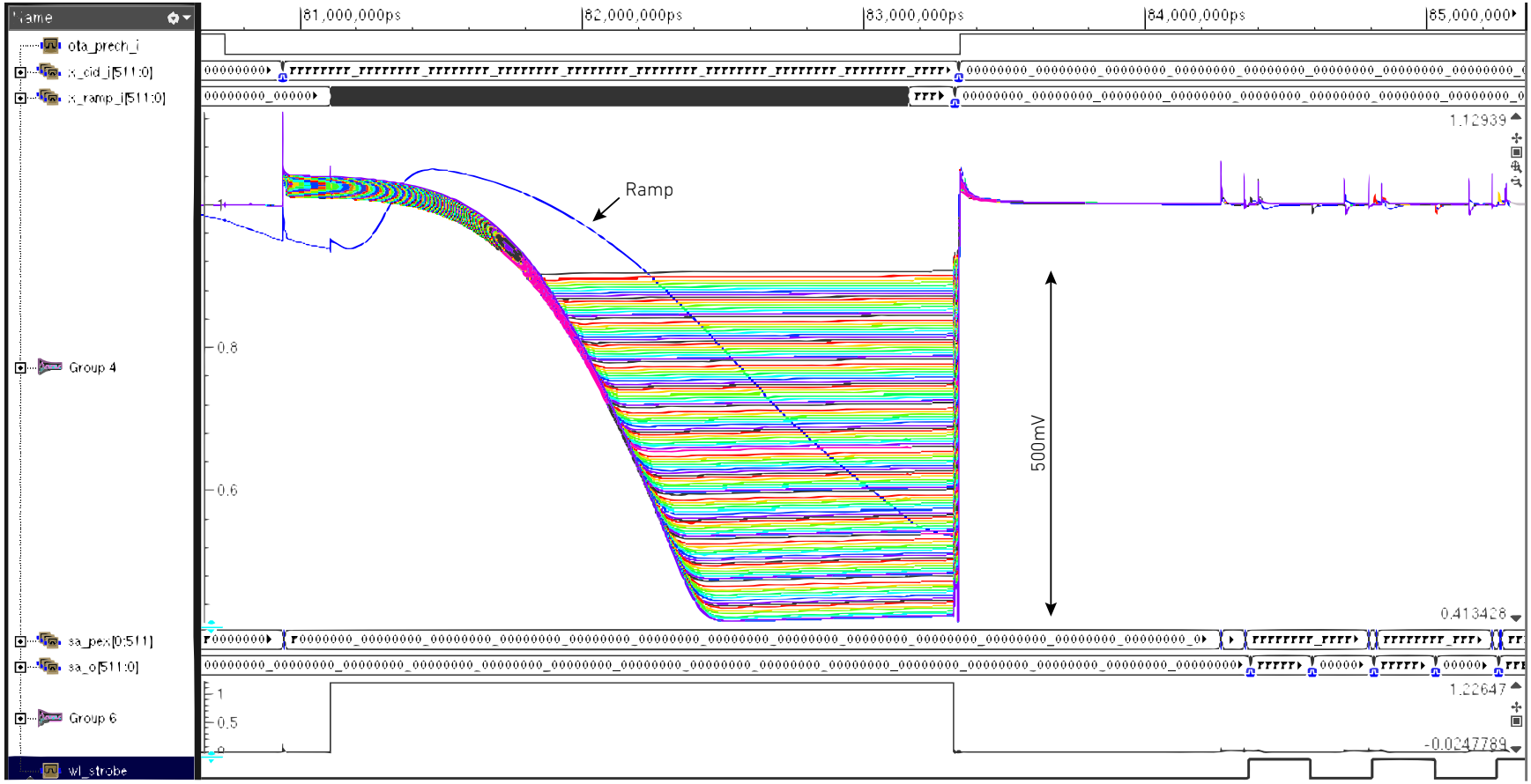


Figure 6.26 *Post-Layout simulation of an entire CID array for a compute and refresh operation.*

to the following observations allowing a reduction in simulation-time from decades to several weeks:

- The key is to not run the full mixed-signal test bench through the AMS simulator. Doing this would result in 150μs of digital simulation which would be merely used for loading memory and configuring parameters; the NoC, MPU, DMA, and Pico-Controller need to be configured and bootstrapped prior to any kind of analog stimulation. Additionally, mixed-signal simulations are not capable of multi-threading the digital portion, therefore an AMS test-bench would run on a single thread, while simulating the full extracted layout on the rest of the cores.

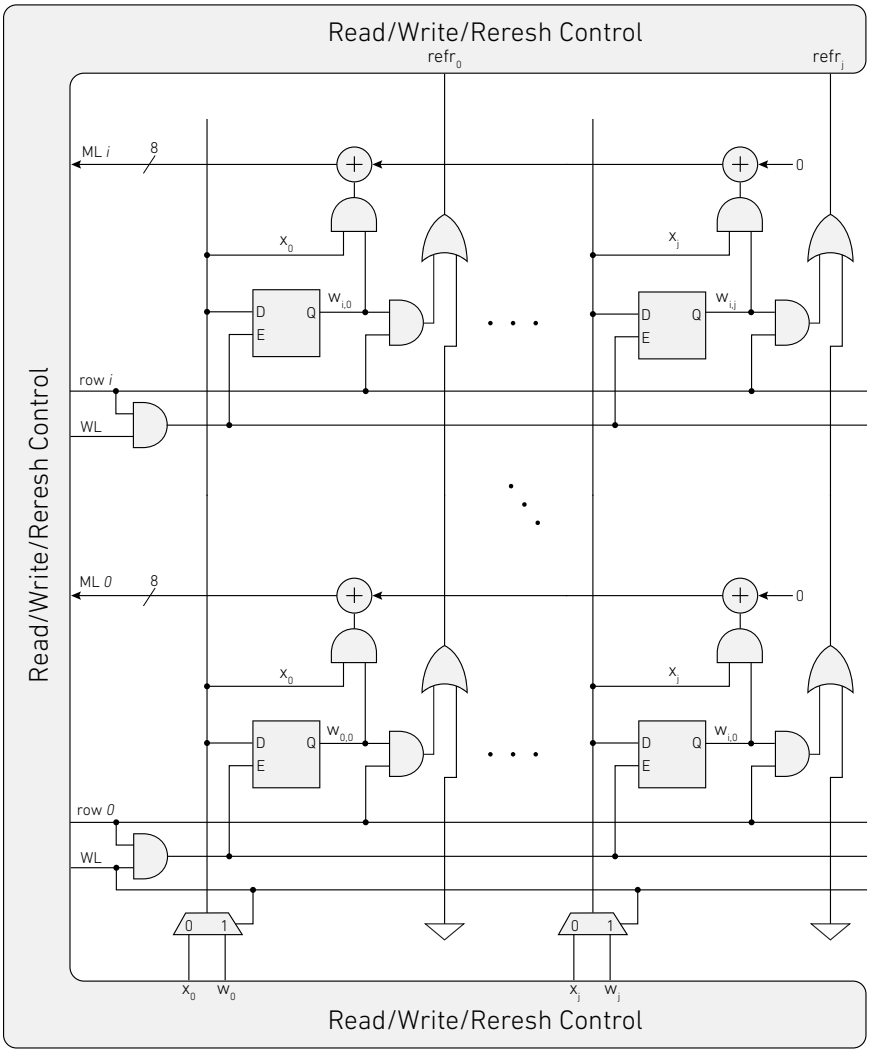


Figure 6.27 *CID-DRAM compute and refresh model used for Post-Layout simulations.*

- A separate pure-digital simulation of the complete sequence (several hundred microseconds) was run in Questasim (MentorGraphics), and value-change-dump (VCD) files were created for the port-interface between the digital and analog blocks.

- The VCD file was used to stimulate the analog block in a stand-alone all-analog Spice simulation in Cadence ADE-XL. Full Spice simulations allow multi-threading, thus resulting in a 24x speed-up of that portion.

- A simulation model was developed for the CID computation and refresh (shown in Fig. 6.27), which was used for validating test benches prior to post-layout validation.

Core Efficiency

Efficiency was calculated based on the fused MAC (FMAC) [78] operation in the same way as for Chapters 3, 4, & 5, where $a \leftarrow a + (b \times c)$ is carried out in full precision in the analog domain, and later re-quantized down to 8-bits with an A/D converter; full specs for the analog core are shown in Table 6.4.

The average power during conversion was calculated by dividing the compute energy by the compute time; a full compute cycle was carried out for 6 rows, supply current was integrated and multiplied by the core voltage, yielding 47.874pJ in a 2μs conversion @ 8-bit output. The conversion time accounts for math-line pre-charging, compute time ($2^N = 256$ cycles), compute hold-time, and return-to-idle state. Therefore, a single row burns an average power of 3.99μW, and accounts for all circuits, including static biases, input capacitor charging, and dynamic power from the comparators. No digital circuits were considered during energy calculations. One row of the CID is capable of computing 16MMACs/s @ 3.99μW of average power, which yields 4TMACs/W.

Table 6.4 *Design specs for the NVM mixed-signal computational unit.*

Technology	65nm CMOS
Core area	0.147mm^2
Core voltage	1.2v
Transistors	278K
Array dimensions	128×512
DRAM memory	8KiB
Bit-Cell area	$1.55\mu m^2$
Input precision	4-bits
Parameter precision	4-bits
Output precision	8-bits
Compute modality	Stochastic
Operation	Fused Multiply-and-Accumulate (FMAC)
Clock frequency	200 MHz
Clock cycles per operation	256
Operations per second (core)	2.1GMACs/s
Operations per second (array)	205.8GMACs/s
Energy Efficiency	4TMACs/Watt

6.5 Conclusion

A Compute-in-Memory (CiM) processor array was designed and fabricated as part of a larger System-on-Chip. The processor was designed in CMOS 65nm technology in collaboration with Northrop Grumman Corporation as a joint effort to advance the state-of-the-art in mixed-signal processors for accelerating vector intensive processing in Machine Intelligence. The processor architecture consists of a 2-layer Network-on-Chip, designed to have a high-speed interface (HSI) as well as a standard General Purpose I/O (GPIO) bus. The processor array is composed of 7×7 tiles, each consisting of a main processing unit, a network interface controller, and a direct memory access controller, which is charge of shuttling data between cores and off-chip communication buses.

The mixed-signal CiM core uses a modified version of the pseudo-DRAM technology described in Chapter 4. Compute gates were separated from bit-lines due to observed parasitic coupling shown to be detrimental to the refresh operation. Charge-based computing has shown to reduce compute-power by operating with voltage swings near the fundamental limits of computation: thermal noise. A novel row-parallel bank of Single-Slope A/D converters (SSL ADC) was designed as the readout stage. The ADC capable of resolving 8-bit accuracies at 2μs per conversion in the sub-pJ energy range, with leakage compensation, and digital non-uniformity correction.

The stringent timing requirements of the compute and refresh circuits pushed the digital controller to a custom instruction set processor, for which a custom assembler was also developed. This allows for users to adapt the controller's stimuli waveforms with arbitrary timing on the array-basis, implying that arrays with different mismatch properties can have tailored control signals. Additionally, the arrays operate under power-aware firmware, which is can enable and disable any bias in the chip on-demand.

Noteworthy, a future improvement to the system would be to consider the use of

cryogenic temperatures ($\sim 80K$) to operate the CiM array. Under these conditions, the lack of leakage and thermal noise could push the array to a more compact and energy efficient design. In this respect, an additional improvement upon density would be to consider an embedded DRAM process, where enormous computational densities could be achieved by using specialized high-density devices, such as deep-trench capacitors.

Additional arrangements were made in order to explore the use of adiabatic charge recycling, a circuit technique that allows the power supply to recover energy that was used in the compute-phase. This operation mode is programmable, yet must operate on whole-chip basis, which makes it just an experimental feature. This limitation was imposed by the number of bond-pads in the die; thus, a future advancement in this respect should incorporate a dedicated pad and on an array basis.

The system was designed to operate on a standard AHB-Lite bus, making it portable and allowing users to instantiate the accelerator on the periphery of any standard microcontroller. As a prime example in this thesis, four CiM cores were attached to the peripheral bus of a standalone S7 SiFive processor. The system was designed and implemented in a separate die in the same technology and shows to be suitable for implantable stimulator devices. Details of the design are shown in Appendix A. All bonded dies mentioned in this chapter are in preparation for testing and measurements will give insight into the future of mixed-signal circuits in edge accelerators.

Chapter 7

Conclusions

This thesis explores the usefulness of unconventional charge-based compute-in-memory architectures that break the Von Neumann memory bottleneck. A scalability and computational bottleneck is especially seen in edge consumer devices, such as smartphones, smartwatches, biometric sensors, etc., where an exponential trend in data growth [86–89] is transmitted wirelessly to remote servers, and subsequently used to perform inference tasks. For this reason, computation is pushed to edge-devices, driving the state-of-the-art in unconventional processing paradigms.

In the realm of unconventional Compute-in-Memory (CiM) processors we discuss two technologies, both involving charge as the means of storage: (i) pseudo-DRAM and (ii) non-volatile primitives. Charge based computing has been an attractive solution since its introduction with charged-coupled device (CCD) imagers in the seventies [71]. Such architectures have been adapted to partake in CiM arrays that have been used for pattern recognition using the same underlying physics [72]. Other work has utilized the same concept in the Charge-Injection-Devices (CIDs) [14, 15, 85] which have also been used for similar pattern recognition tasks. However, these computing elements have not been implemented with feature sizes smaller than 180nm, until now.

In this work, the Charge-Injection-Device (CID) concept is taken to smaller CMOS 65nm & 55nm technology nodes, and has shown promising results in its use

as a computational primitive in edge-computing applications. Power efficiency is achieved by operating near the fundamental limits of KTC thermal noise. It has also been shown that the limiting factor in precision was dominated by device mismatch. Thermal noise and leakage have proved to be problematic but could be mitigated easily by operating the chips at cryogenic temperatures (for specific applications). In this scenario, the concept of multilevel storage in the charge-domain could be explored, as well as further reduction in energy due to the lack of necessity to refresh memories. Even at these temperatures, the mismatch problem should be addressed by introducing larger capacitance (and higher density) devices such as deep-trench capacitors. Additional novel circuits were developed to increase the computational density and drive power numbers into the pJ/MAC (8-bits, fixed point), providing energy efficiencies comparable to other state-of-the-art unconventional processors, such as IBM's TrueNorth. Non-Volatile primitives were also explored for their use in CiM architectures, where computation took place in the charge-current domain, near the physical limits of shot-noise, providing similar efficiency and computational density as its pseudo-DRAM counterpart.

Both DRAM and NVM provide high density memory crossbars, yet the peripheral overhead circuits are very different. NVM has proven to be cumbersome to work with due to the increased complexity and overhead of peripheral high voltage circuits; approximately 85% in die area is utilized for high voltage control circuits, which in turn have proven to be easily ruptured if not operated adequately. Conversely, DRAM crossbars require only a small overhead in control circuits ($\sim$5-10%), and do not demand special high-voltage process options. Moreover, the DRAM cells designed in this work have been embedded in a CMOS logic process, resulting in no extra cost for special devices such as deep-trench capacitors found in commercial DRAM.

In this work, models of neuromorphic electronic circuits were emulated in FPGAs, and later deployed and tested in silicon within larger Systems-on-Chip, supported

by conventional co-processors and high-bandwidth interfaces. They have proven to be a cost-effective solution to solve computations within the realm of machine learning, both from the device and architectural level. More specifically, Vector-Matrix Multiplications (VMMs) have been optimized, where computation was embedded into local memory, storing data in a unary domain; unary coding techniques such as Pulse-Width-Modulation (PWM) and event-based stochastic modalities were exploited, showing to be resilient to interference at lower post-processing cost/complexity when compared to binary coding.

The main downside of mixed-signal computational-memories lurks in the A/D and D/A conversions, which are necessary steps in the process. The energy expenditure in these conversions may overhaul the energy efficiency of the compute-cores. In this work, computational efficiencies were always calculated based on the existing compute hardware. Chips described in Chapters 3 & 4 did not include A/D converters, therefore conversion power was not reported. However, Chapters 5 & 6 do report power numbers including A/D converters (since they were indeed embedded in the the arrays). Having mentioned this, the computation efficiency across chips was reported for the same operation: the Fused Multiply-and-Accumulate (FMAC), where inputs and parameters are represented in 4-bit fixed point arithmetic, and full-precision outputs are downsampled to 8-bits using A/D converters.

Future work includes testing of two recently fabricated Systems-on-Chip: (i) the Suanpan tabled, which was re-fabricated due to issues in the I/O connections, and (ii) the Edge-Super-Computing (ESC) array, which includes the most recent work reported in this thesis. Additionally, it would be of great interest to adapt commercial DRAM to perform in-memory computing, which is a task that to our knowledge has not yet been shown in literature. Commercial DRAM requires fewer metal masks than logic processes, which may offset the cost in the additional processing steps to produce specialized capacitors. It is not clear why commercial DRAM has not yet

been used in experimental CiM blocks but shows clear potential.

Predicted results from the ESC-chip are actively being used for benchmarking state-of-the-art processing for artificial intelligence applications in low-power CMOS technology nodes, such as 22nm and 16nm. The main conclusion of the thesis is that the compactness and efficiency of Quasi-Digital computing arrays are indeed sufficient to offset the lurking D/A and A/D converters, as long as a low-precision fixed-point arithmetic is considered. The main question to be answered is whether analog and mixed-signal circuits can keep up with scalable digital unconventional processors, such as Intel's Loihi and IBM's TrueNorth.

Appendix A

Embedded Neural Accelerator Suitable for Implantable Devices

This thesis has explored neural engines for embedded applications by leveraging specialized compute-in-memory cores that perform data intensive tasks. In the contemporary world, embedded microcontroller platforms such as the Arduino [90] and the HiFive [91] are gaining popularity due to their portability and low power for resolving general purpose I/O control. Nonetheless, they are not suitable to perform high-volume/high-throughput cognitive computations due to their lack of specialized neural engines, and their bulky size make them a non-viable option for neural implants. Most implantable devices are paired with a microcontroller which is charge of housekeeping and controlling the current-mode stimulators, making them an essential part of the system. Moreover, vector processing is a common task in implants, such as matrix translations in gyro data used in vestibular-ocular-reflex stimulation. In such systems, the physical separation between the stimulator and the microcontroller raises a safety concern, thus if connection is lost, dire consequences may occur.

In this work, the SiFive S7 platform [92] is taken as a design start-point and expanded to include a quad-core Compute-in-Memory (CiM) edge accelerator located on its periphery. The System-on-Chip (SoC) is suitable for experimenting with

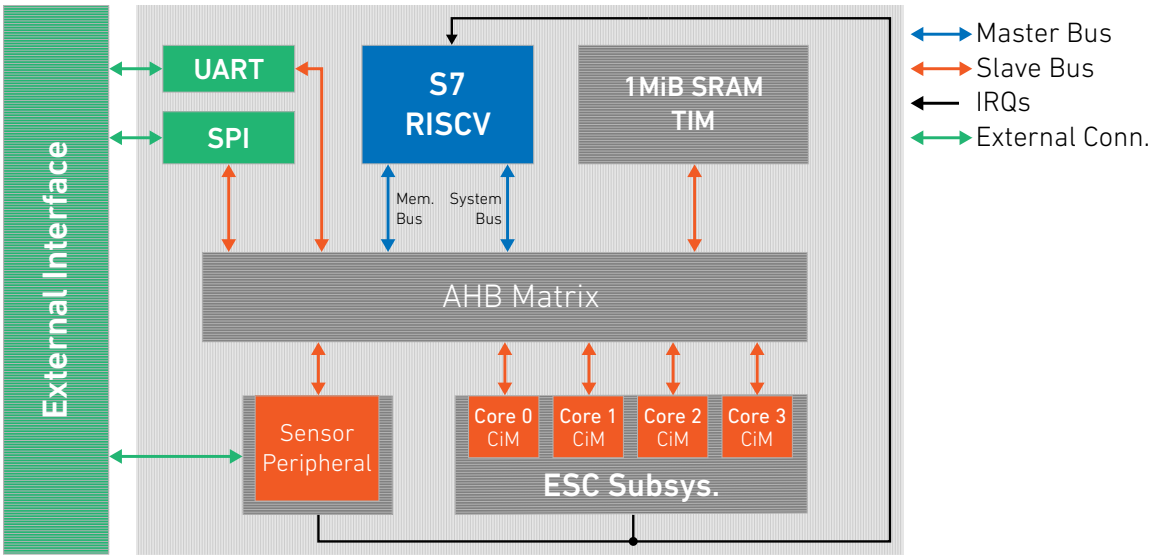


Figure A.1 *System-on-Chip block diagram*

implantable devices and was fabricated in a CMOS 65nm technology. The chip was bonded and is in preparation for testing.

A.1 S7 CiM Architecture

A descriptive diagram of the SoC is depicted in Fig. A.1. The figure shows the system's main blocks: an S7 processor, a quad-core CiM accelerator, memory, and communication peripherals. In particular, a sensor peripheral was designed by Jonah Sengupta and utilizes a special asynchronous interface that allows seamless connection to a Dynamic-Vision Sensor (DVS), which was fabricated in a neighbor die. The design of the DVS and sensor peripheral escapes the scope of this thesis and will not be described in further detail.

S7 Core

The S7 core RTL was supplied by SiFive[1] through a university license. SiFive provides an online portal allowing users to customize their processors on-demand. The configuration details used in this work are depicted in Fig. A.2, showing a 64-bit

[1] `https://www.sifive.com`

Figure A.2 *SiFive S7 core and configuration details.*

RISCV processor with a floating point unit and 2× 32KiB Instruction/Data cache. It supports both local and global interrupts, each controlled by the Core-Local Interrupt Controller (CLINT) and the Platform-Level Interrupt Controller (PLIC), respectively. It includes physical memory protection (PMP), a debug unit, and 1MiB of cached memory space (L2) providing low latency memory access. The core also has accepts connections to multiple peripherals (System Port, Peripheral Port, and Front Port), each of which can independently support either the TileLink or AMBA protocol. SiFive provides a TileLink to AHB protocol conversion block, which allowed seamless connectivity between the CiM accelerator and the processor's control and memory system.

The S7 is bootstrapped via Read-Only-Memory (ROM), which holds 8 bootloaders. The CPU chooses which routine to execute based on external configuration pins. The

boot sequence consists of the CPU interfacing with a peripheral which allows an external program to be copied to RAM. After the bootload process is takes place, the program counter is set to point the base address of the newly loaded code and execution continues.

CiM Quad-Core

An Edge Super-Computing (ESC) Subsystem was designed by tiling four of the CiM cores that were previously designed for a larger, scalable SoC. The CiM unit is a digital-input, analog-compute, digital-output edge accelerator, with 8KiB of pseudo-DRAM computational-memory, a custom controller, a standard AHB Bus, and a handful of configurable IRQs. The CiM unit is described in detail in Chapter 6.

UART

The UART peripheral is used to shuttle data in and out of the chip and interfaces with the AMBA BUS via a UART-to-AMBA converter. The UART frequency runs at a fixed division of the base CPU clock speed. The UART has a simple control scheme, where if a destination address falls in the UART target space, a parallel-to-serial state machine is triggered. Polling the UART requires reading from its corresponding base address, which returns 64 bits of which the lowest 9 bits are relevant. The most-significant of those 9 bits indicates whether the data is valid. A value of 0 indicates that the data is ready, which means that the lower 8 bits contain the data. Nonetheless, using an interrupt service routine (ISR) to read the data is simpler from the perspective of interacting with the UART because once the ISR is properly triggered, it is known that the data is already valid and stored in the FIFO. Therefore, the ISR can simply read from the base address and extract the lowest 8 bits from that 64-bit value returned on the bus.

SPI

The SPI peripheral is composed of four parallel SPI devices, each one sharing the same *chip-select*, *sck*, *instruction*, and *address*; in this sense, it is a quasi-QSPI bus. Each lane has its own separate byte of data that it can read or written in parallel, which allows for 32-bit transfers during each transaction.

Reading from the SPI proceeds as follows. First, the command and address must be established. The command is an 8-bit value and the address is a 24-bit value. Both of these values are concatenated into a single 32-bit register where the command comprises the highest 8 bits and the address is the lower 24 bits. The instruction value to read is 0x03, and the address depends on where in the SPI device the data should be read from. That value must be written to the SPI_INSTRUCTION_ADDRESS register in order to initiate the read. When the read completes, the SPI read IRQ goes high. So if the interrupt is configured properly, an ISR can be used to clear the read interrupt. In order to poll the SPI, first the instruction and address are set and written. Then the SPI_READ register is polled while the read IRQ remains 0. Once the IRQ becomes 1, the SPI_READ register is polled again while the read IRQ remains 1. This ensures that the SPI peripheral is in sync with the CPU and the SPI has had time to successfully deassert the IRQ before moving on to the next command. **Writing** proceeds in a similar manner to reading, except that the instruction to write is 0x02.

Bootloaders

There are a total of 8 bootloaders for each CPU, providing multiple options for programming the cores for the sake of redundancy as well as flexibility. These bootloaders span three peripherals, namely the Config (CFG) and Data UARTs as well as the quad-channel parallel SPI interfaces. The full memory space for each CPU here is split into two equal and contiguous halves called “Mem 0" and “Mem 1". Below

are the possible boot options for each CPU core:

1. UART CFG into Mem 0
2. UART CFG into Mem 1
3. UART Data into Mem 0
4. UART Data into Mem 1
5. SPI into Mem 0 on 4 channels
6. SPI into Mem 1 on 4 channels
7. SPI into Mem 0 on 1 channel
8. SPI into Mem 1 on 1 channel

Options 1-4 are very similar to one another in that they all load data from the two UART pins. The differences are that 1 and 2 read from the CFG UART port and place the data into the beginning of Mem 0 or Mem 1, respectively. 3 and 4 read the data from the Data UART port and also place the data into the beginning of Mem 0 or Mem 1, respectively. All four of these UART bootloaders read a variable amount of data into memory which is specified as the first three bytes of the data sent to the UART. This amount typically indicates the size of the program to be stored in memory and later executed, but this can also include extra data that is to be used by the program after the contents of the executable are placed in memory.

Each of the UART bootloaders begins by sending the value 0x55 to let the outside world see that the chip is alive and ready for incoming data. The value 0x55 also provides a simple way to gauge the baud rate on an oscilloscope because it looks like a square wave.

Options 5-8 are different. Instead of specifying the amount of bytes to read at the beginning of the data, these SPI bootloaders read a fixed amount of data each time.

Options 5-6 read the data across all 4 channels, each channel reading one byte at a time, so 32 bits are read at once. Options 7-8 read the data on only one channel, so 8 bits are read at once. Therefore, options 7-8 take 4 times longer to read all the data than options 5-6, but they allow the processor to boot by only using one external SPI flash memory device instead of having to configure four of them.

There are three bootloader configuration-pins in the top-level design that go down to the ROM which select the desired bootloader choice. The code that creates the Verilog file describing the ROM includes the code for choosing the bootloader, which minimizes the chance of mismatch between the ROM itself and the code to choose the offsets within that ROM for the CPU's starting boot address. This boot address is sent back out of the ROM file and into the CPU so that the CPU knows which location to boot from in the ROM.

Test Chip S7 Memory Map

Table A.1 shows the base addresses of the memory map for the S7 test chip architecture. The test chip has twice as much memory as the main chips have for the S7, but the bootloader is unchanged. Therefore, Mem 0 is the same size and Mem 1 must contain the rest of the address space, since the Mem 0/Mem 1 dividing point is determined by the bootloader.

Peripheral	**Base Address**
ROM	0x40000000
UART CFG	0x41000000
UART Data	0x42000000
SPI	0x43000000
Sensor	0x44000000
ESC Subsystem	0x45000000
Mem 0	0x80000000 (256 KiB)
Mem 1	0x80040000 (768 KiB)

Table A.1 *Test chip S7 memory map.*

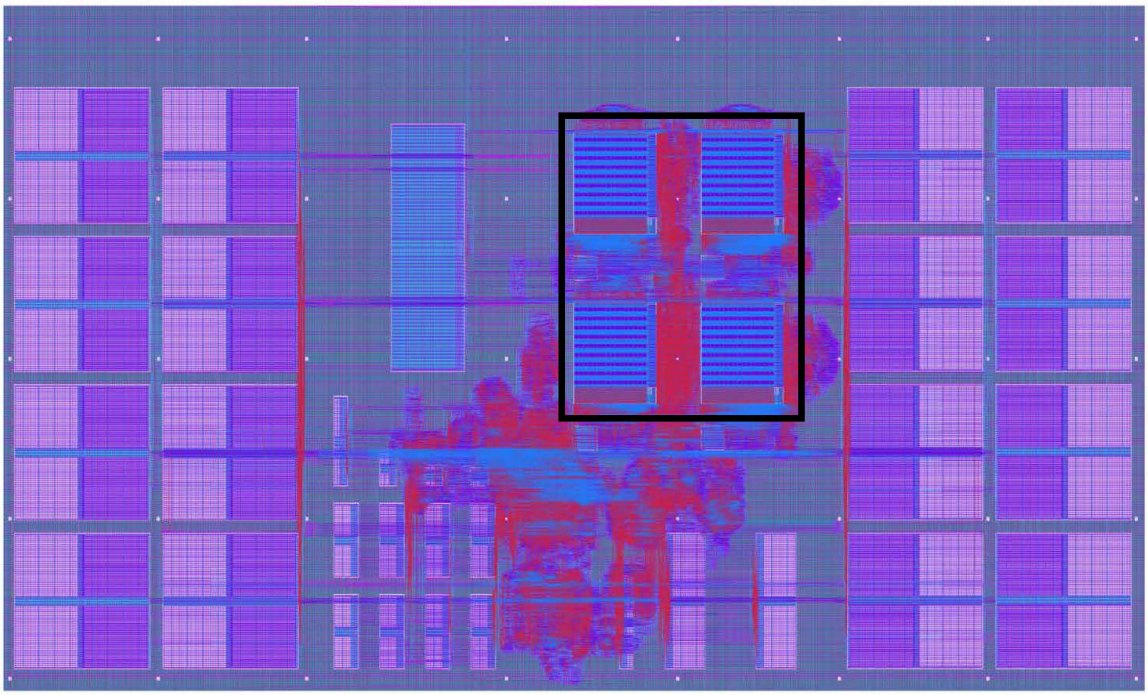


Figure A.3 *SiFive S7 processor with a quad-core AI accelerator.*

A.1.1 Implementation

Figure A.3 shown the S7 core and accelerator layout. The quad-core custom accelerator is shown in the black box in the middle of the layout, while the surroundings show the S7 synthesized layout and memory. The chip is approximately 7×4 mm, and was fabricated in a 65nm technology.

A.1.2 Conclusion

A SiFive S7 processor was used to create microcontroller with an embedded neural engine on the peripheral bus. The amount of memory used (1MiB) is suitable for running typical Arduino-style programs and the addition of the neural CiM accelerator opens the door to solve data-intensive vector processing applications. Since the system is monolithically integrated, it is especially suitable for implantable devices, where local processing is of great interest due to the reduction of energy consumption in their wireless links. A suitable target application for this SoC would be a vestibular

implant, which typically requires an external processor reading continuous data from gyros; if external communication is lost between the stimulator chip and the wireless link (external), vertigo sensation could have dire consequences to the user. If the gyros are integrated in the stimulator package, the CiM accelerator could perform real-time processing on acceleration data; with the help of the S7 processor no need for an external link is required to operate. Noteworthy, a neural stimulator would require a current-mode (DAC) and a wireless link in order for in order for it to be implantable-ready.

A.1.3 Acknowledgment

The author would like to thank Daniel Mendat, Jonah Sengupta, and Martin Villemur for their contributions in designing and documenting the integrated circuit described in this Chapter.

Appendix B

Neuromorphic Self-Driving Robot Platform with Spike-Based Vision and Decision-Making

Kate D. Fischl[*‖], Gaspar Tognetti[*‖], Daniel R. Mendat[*†], Garrick Orchard[§], John Rattray[*], Christos Sapsanis[*], Laura F. Campbell[*], Laxaviera Elphage[¶], Tobias E. Niebur[*], Alejandro Pasciaroni[‡*], Valerie E. Rennoll[*], Heather Romney[5], Shamaria Walker[5],and Andreas G. Andreou[*†]

[‖]Contributed equally

[*]Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, Maryland 21218

[†]Center for Language and Speech Processing, Johns Hopkins University, Baltimore, Maryland 21218

[‡]IIIE - CONICET/UNS, Bahía Blanca, Argentina

[¶]Western High School, Baltimore, MD 21209

[§]Temasek Laboratories and the Singapore Institute for Neurotechnology at The National University of Singapore, Singapore

Email: {gtognet1, kfischl1, dmendat4, csapsanis, jrattra1, apascia2, andreou} @jhu.edu garrickorchard@nus.edu.sg

B.1 Abstract

We present our work on a neuromorphic self-driving robot that employs retinomorphic visual sensing and spike based processing. The robot senses the world through a spike-based visual system – the Asynchronous Time-based Image Sensor (ATIS) – and processes the sensory data stream using IBM's TrueNorth Neurosynaptic System. A convolutional neural network (CNN) running on the TrueNorth determines the steering direction based on what the ATIS "sees." The network was trained on data from three different environments (indoor hallways, large campus sidewalks, and narrow neighborhood sidewalks) and achieved steering decision accuracies from 68% to 82% on development data from each dataset.

B.2 Introduction

Neuromorphic hardware departs from the sequential processing of Von Neumann architectures by using a massively parallel design to abstract biological principles [93–95]. The IBM TrueNorth Neurosynaptic System [48, 49] is a portable hardware architecture that is well suited for autonomous system applications or other real-time, power constrained environments. The TrueNorth chip is also a good choice for spike-based machine learning implementations due to its spike-based computational abilities. Deep learning [96] with convolutional neural networks (CNNs) is particularly suitable to this platform because of IBM's energy-efficient deep neuromorphic networks (Eedn) software framework for training the hardware to efficiently run these networks with the hardware constraints posed by the chip [97].

B.3 Background

Machine learning is a growing field used in a wide range of applications [98–100]: from customizing web browsers by learning user interests to automatic speech recognition, stock market prediction, image classification, and more recently autonomous driving vehicles. In some machine learning techniques, human extracted features are processed with machine learning algorithms. The endeavor of feature extraction requires a great deal of time and effort on behalf of trained engineers. This tedious and time-consuming exercise inherits the "human bias" which may result in neglecting important features which could be critical to system performance. This effect may be mitigated using *deep learning*, [96] in which new data representations are automatically learned, thus enabling computer algorithms to interact with raw data while maintaining system performance.

Pomerleau introduced an autonomous land vehicle in 1989 [101] that utilizes a 3-layer neural network, making it one of the first machine learning approaches in

autonomous navigation. More than a decade and a half later, the Defense Advanced Research Agency (DARPA) introduced the DARPA Autonomus Vehicle (DAVE) [102], which demonstrated autonomous navigation in an alley of obstacles without communication to a home base. The system consisted of a 6-layer convolutional neural network (CNN) and was trained from end-to-end by mapping raw images, taken while the system was driven by a human, to steering angles. A more recent approach, DAVE-2, was proposed by an NVIDIA research group [100]. Interestingly, their vehicle was able to navigate areas under different weather conditions and with limited visibility. Their 9-layer CNN managed to detect features suitable for predicting vehicle movement. During the 2016 Telluride Neuromorphic Engineering workshop, a team lead by Prof. Jeffrey Krichmar worked on autonomous navigation using their Android based robot platform with vision sensing and pre-processing on an Android phone and a deep CNN implemented on IBM's TrueNorth Neurosynaptic System [103]. In the same workshop, we have worked towards a neuromorphic robotic system that employs a retinomorphic vision system, the ATIS sensor [104] followed by processing on the TrueNorth chip; details of the work on the hardware and software architecture as well as some preliminary results are presented in this paper.

B.4 Methods

B.4.1 Robot Platform Hardware

The robot platform (see Fig. B.1) was built using guidelines from the University of California, Irvine (www.socsci.uci.edu/∼jkrichma/ABR/) and consists of a 6WD aluminum differential drive chassis propelled by 6 motors using 75:1 gearboxes. The chassis payload allows all data collection hardware to be easily mounted. The motors are controlled by a 2x30A controller which accepts drive and steer commands from an Android smartphone mounted on a pan and tilt system. This phone runs a host application, which in turn receives commands from a client tablet running a

controller application. The former is adapted from *Android-Based Robotics* code (https://github.com/UCI-ABR) and provides discrete driving commands, i.e. discrete speed and discrete turning angles. This approach allows the robot to maneuver in tight spaces as well as open areas while providing clear, unambiguous driving commands (labels) for training a CNN. Finally, the host phone communicates with the motor controller via a special development board, specifically a SparkFun IOIO micro-controller, designed to interface with Android platforms. Figure B.1 shows the front of the chassis supporting a pan servo-motor, which holds the ATIS.

ATIS is a custom CMOS vision sensor with 304×240 pixel resolution [104]. Each ATIS pixel operates independently, detecting changes in its own log-illumination, and outputting an *event* whenever such a change is detected. An *event* consists of the pixel address and 1 bit of polarity to indicate whether the change was an increase or decrease in log-illumination. These *events* are communicated off-chip with very low latency, and therefore the time at which an event is output by the ATIS can be considered as the time at which the corresponding change in log-illumination was detected.

The ATIS is controlled by a Xilinx Opal Kelly XEM6010-LX45 board, which allows

(a) (b)

Figure B.1 *Robot data collection platform: front (a) and side (b) views. Notice the ATIS mounted on the front, the smartphone in the middle, and the Surface Pro mounted on the motor controller box.*

for logging of event based data onto a Microsoft Surface Pro. During data collection, in addition to storing the ATIS data on the Surface Pro, the motor pulse-width modulation (PWM) control signals and photos taken from the Android smartphone of the scene are stored on the Android phone. These data are processed later to form the training input for the deep CNN.

B.4.2 Real-Time Autonomous Platform

The real-time system operates similarly but does not require the Surface Pro since the TrueNorth chip can communicate directly with the ATIS sensor with spikes and there is no need to log data. In this system, the ATIS spikes are sent to a portable 12V-powered Linux PC mounted on the robot which is powered with another battery. This PC simply groups spikes together as described in Section B.4.5 and sends these batches directly to the TrueNorth hardware using TCP communication. The TrueNorth pushes the spikes through the trained network stored on board and then outputs the results back to the portable PC. Finally, the PC decodes these results and commands the motor controllers to move the wheels in the appropriate fashion. Figure B.2 shows how information flows using this platform. The autonomous system bypasses the Android phone and tablet which were previously used to control the robot when gathering training data since these devices are not required when the robot drives by itself.

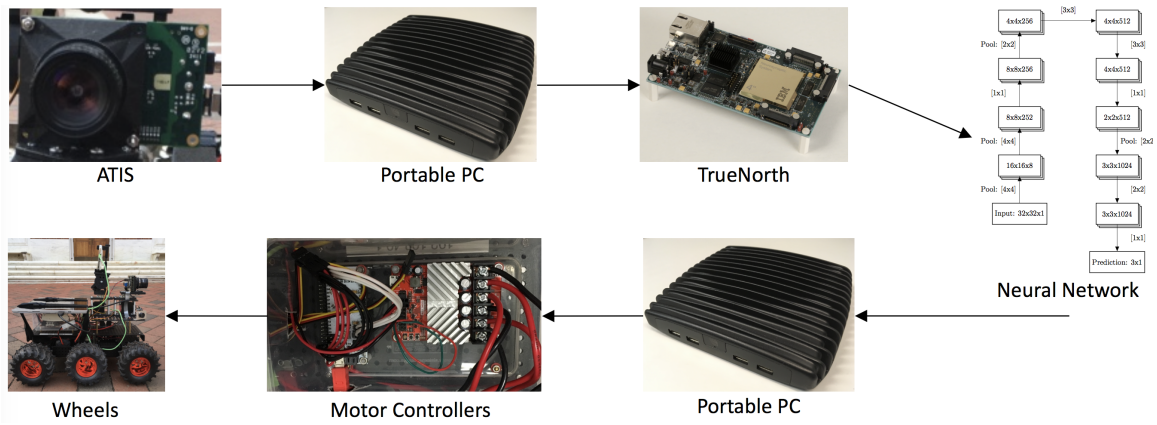
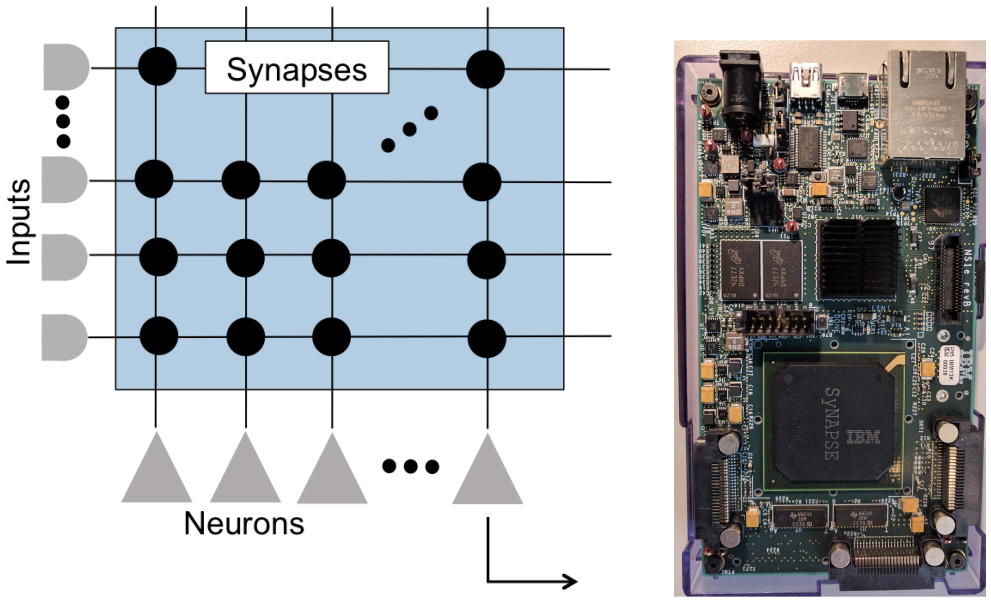


Figure B.2 *Data flow in the neuromorphic robot system.*



(a) (b)

Figure B.3 *(a) A basic TrueNorth core. (b) The NS1e evaluation platform, containing the TrueNorth chip.*

The TrueNorth architecture is a low-power, massively-parallel chip multiprocessor containing over 1 million independently configurable neurons and over 268 million individually programmable synaptic connections. The chip consists of 4096 cores, each of which contains 256 neurons and 256 axons connected in a programmable crossbar array. All neurons have parameters (leak, threshold, synaptic weights, etc.) programmable by the user prior to run-time. This programming model provides great flexibility and also complexity given that each neuron, synapse, and axon must be individually coded. Fortunately, IBM has created a system to make building, training, and running deep convolutional networks on the TrueNorth hardware simpler than programming everything from scratch (see Section B.4.5). A basic TrueNorth core and a picture of the NS1e evaluation platform, on which the TrueNorth chip is mounted, are illustrated in Fig. B.3.

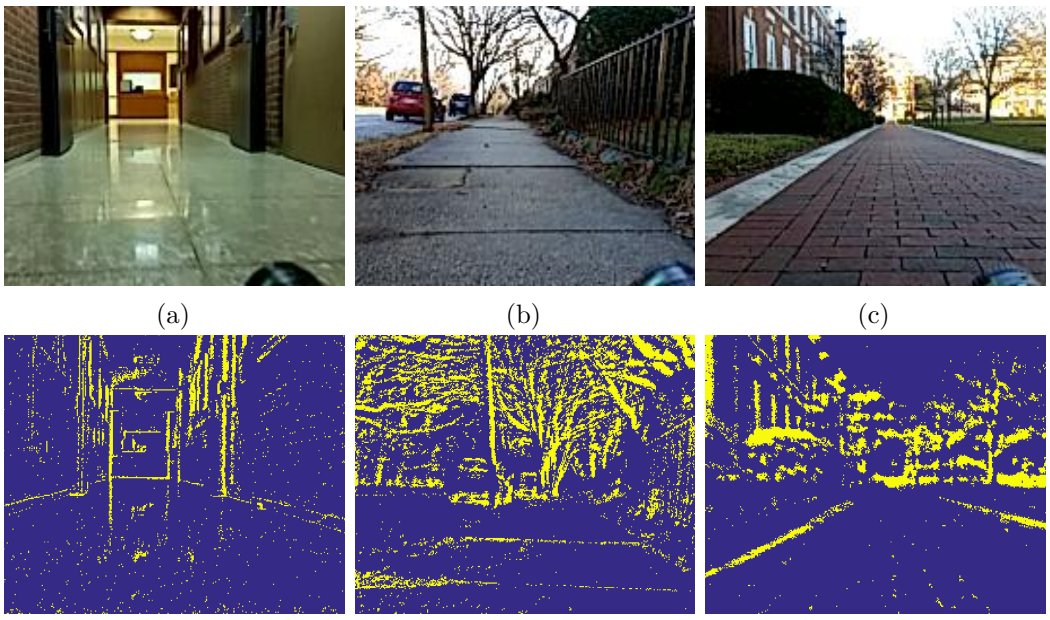


Figure B.4 *Examples of the three collected data sets. (a) Data collected using the Android phone while the robot was driven in the hallways of a building at Johns Hopkins University (d) and the corresponding input to Eedn. (b) Data collected using an Android smartphone while the robot was driven on a residential sidewalk and (e) the corresponding input to Eedn. (c) Data collected using the Android smartphone while the robot was driven on a campus sidewalk path and (f) the corresponding input to Eedn.*

B.4.3 Data Collection

Data was collected from three different environments including the hallways of a building at Johns Hopkins University, campus sidewalk paths (approximately 2 meters wide), and residential sidewalks (approximately 0.6 meters wide). Figure B.4 shows images taken using the Android smartphone camera taken during data collection and the processed ATIS output.

B.4.4 Data Pre-Processing

The main challenge in data pre-processing was time-domain alignment between asynchronous-time address events and uniformly sampled (usually 30Hz) driving commands from the host smartphone. Due to the different nature of the recording systems, time stamps were aligned offline using a custom algorithm, explained below.

On the smartphone data side, data were truncated to the onset of movement which was found by identifying peaks in the first derivative of the robot speed, derived from raw PWM signals. On the ATIS side, the onset of movement was captured by searching peaks in the derivative of the inter-spike-intervals (ISI). When the robot is motionless, spikes are rare, mostly due to noise or changes in light. Conversely when the robot is moving, the time-domain differences in the scene relative to the robot create a high firing rate, thus a lower inter-spike-interval. As a result, the first major peak in the derivative of the inter-spike-interval reveals the onset of movement. After the data alignment, spline interpolation was used to append labels to the AER data.

B.4.5 Network Training

After the ATIS output spikes were assigned labels, they were then used as input to the training of a deep learning neural network trained to output the correct driving direction (left, right, center). Training was performed using IBM's energy-efficient deep neuromorphic networks (Eedn) software framework for TrueNorth [97]. Eedn trains convolutional networks, whose connections, neurons, and weights have been constrained to map onto the TrueNorth chip. Training produces a corelet, or composable hardware description, that can be programmed directly onto the TrueNorth chip [50]. Eedn allows the user to specify the parameters of each layer within the network, including number of features, kernel sizes, padding, stride size, learning rates, and many others. While training the network, Eedn can train and test either within "Compass," a 1-to-1 scalable software simulator, or directly on the TrueNorth chip [51].

Because the ATIS generates spikes in very fine-grained intervals, to train the network in Eedn spikes are sent into the system in batches, similar to the way spikes are delivered during real-time communication. Spikes were aggregated within a specific time window to generate spike-frames. Each of these groups (frames) was assigned a label based on the majority of occurrences within that time frame. Once the spikes

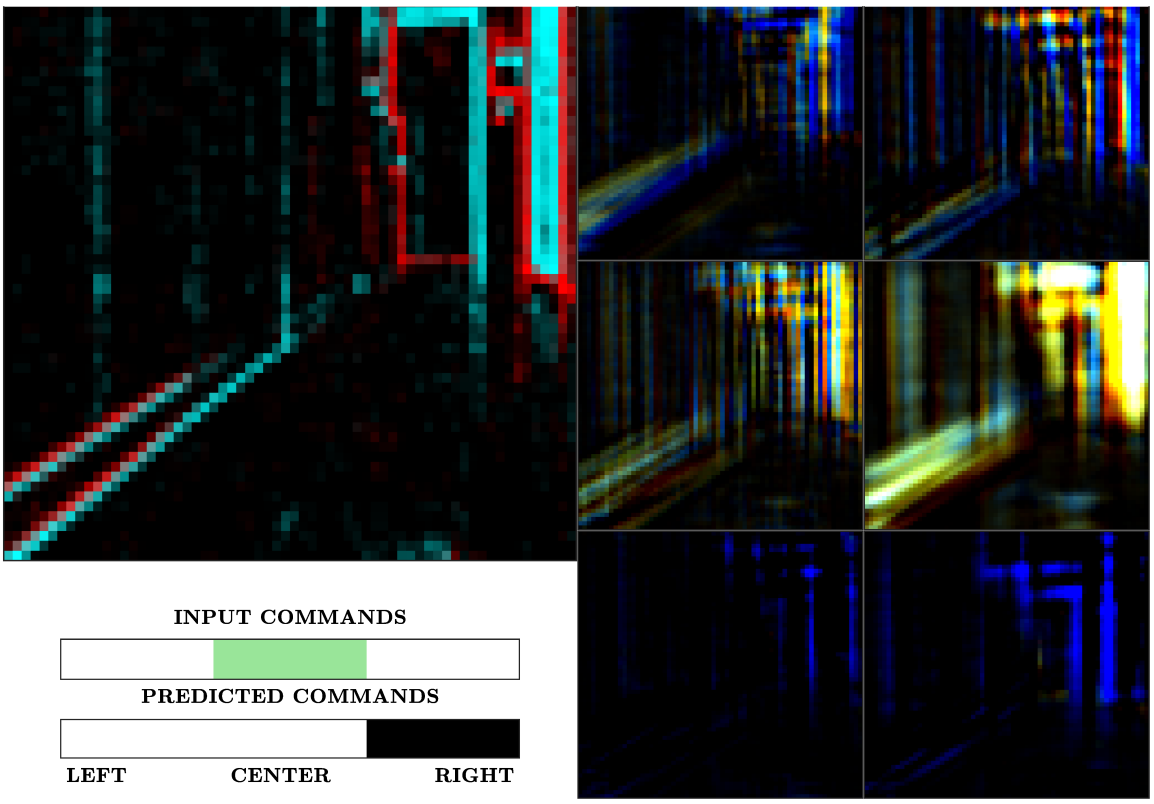


Figure B.5 *MATLAB simulation results for CNN tested on ATIS frames. Note the output of the 6 convolutional filters appear to be performing edge detection. Moreover, while the driver is aiming at the wall, the predicted command indicates a correction towards the right.*

were grouped and labeled, they were divided into *train*, *development*, and *test* sets. The training set consisted of roughly 60% of the collected data, the development set was roughly 20%, and the test set was also roughly 20%. The hallway dataset contained 6,591 training groups, 2,242 development groups, and 2,189 testing groups. The campus sidewalk dataset contained 18,076 training groups, 6,099 development groups, and 6,071 testing groups. Finally, the residental sidewalk dataset contained 12,865 training groups, 4,347 development groups, and 4,271 testing groups.

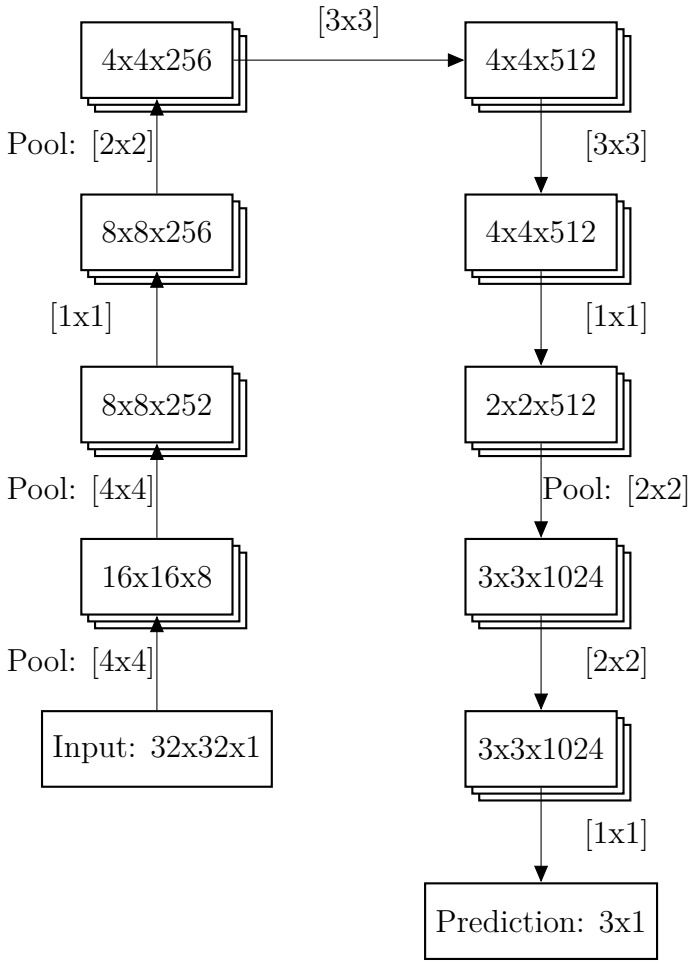


Figure B.6 *Convolutional Neural Network implemented on the TrueNorth hardware and trained in IBM's Eedn framework. Each block shows the size of the output at each layer and the connections between layers are labeled with the size of the kernel used in brackets.*

B.5 Results

B.5.1 Preliminary Results

Preliminary work included training a CNN on GPUs using the MATLAB Neural Network Toolbox. Two different datasets were used to train the same network. The network architecture included 1 convolution layer with 6 5x5 filters with a stride of 1, 1 2x2 max pooling layer, 3 fully connected layers, and 3 outputs corresponding to the labels *left*, *center*, and *right*. Intuitively the convolutional layers will do feature extraction while the fully connected layers carry out the control, however it is not

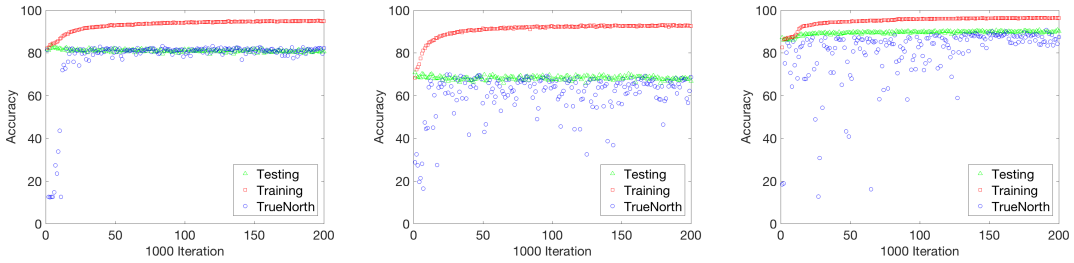


Figure B.7 *Results from training a deep neural network in IBM's Eedn software framework using the (a) residential sidewalk dataset, (b) hallway dataset, and (c) campus sidewalk dataset.*

possible to draw a definite line between the two tasks. The first dataset was comprised of 45 minutes of 40x40 pixel smartphone images, down-sampled to 10 FPS in grayscale. 80% of this data was used for training, and testing on the remaining 20% produced an accuracy of 79%. The second dataset consisted of 35 minutes of ATIS output data that was down-sampled into 64x64 pixel frames sampled uniformly at 10 FPS. The ATIS output polarity was used to create a 3 value color image in MATLAB, akin to RGB. This dataset was again trained on 80% of the data and tested on 20% which produced an accuracy of 81%.

Figure B.5 shows an input image taken from the test set and pushed through the trained network. The bottom of the figure depicts driving input and predicted commands, and the sub-figures on the right illustrate the output of the 6 filters from the convolutional layer. Intuitively it can be seen that the network is carrying out edge detection. In this particular image, the user is driving the robot left towards the wall and the CNN is predicting a correction towards the right. During data collection the robot was randomly steered toward the wall and then corrected to increase the variability of the data and to teach the robot to recover from mistakes.

B.5.2 Simulation Accuracy

After showing the viability of this CNN approach in Section B.5.1 a separate network was trained in Eedn (see Section B.4.5) for use on the TrueNorth hardware. The network that gave the highest accuracy on the development datasets contains 11 layers. This network is shown in Figure B.6. The first layer is a data layer which sends information into the system. Next, there are three sets of layers, each set containing three individual layers. The first set contains two pooling layers followed by a convolutional layer. Each of the second and third sets contain a pooling layer followed by two convolutional layers. The final standard layer is the prediction layer that chooses the direction the robot should be steered, and finally there is a single loss layer (not pictured in Fig. B.6) that Eedn uses to train the network.

This network was trained for 200,000 iterations on each of the three datasets. When trained on the residential sidewalk dataset, this network yielded a training accuracy of 95%, a development accuracy of 81%, and an accuracy of 82% when run directly on the TrueNorth hardware. The hallway dataset resulted in a training accuracy of 93%, a development accuracy of 67%, and an accuracy of 69% on the TrueNorth hardware. The campus sidewalk dataset gave a training accuracy of 96%, a development accuracy of 90%, and an accuracy of 87% on the TrueNorth hardware. The accuracy on the TrueNorth hardware and the development data differ due to the constraints placed on weight values in the actual hardware. The training and development curves for each dataset can be seen in Fig. B.7.

B.6 Conclusion

We have shown the construction of a fully neuromorphic autonomous robotic driving platform including a spike-based vision system and a spiking “brain" that successfully processes that sensory information. Even with limited training data the robot deter-

mines the correct direction to steer up to 82% of the time depending on the dataset. We expect those results to improve further with more training data and extended training, and these results will be fed into a successful real-time implementation.

Acknowledgment

The authors would like to thank Rodrigo Alvarez-Icaza and Andrew Cassidy of IBM for their support. This paper was partially supported by NSF Grants INSPIRE SMA 1248056 and 1540916: SL-CN Cortical Architectures for Robust Adaptive Perception and Action through the Telluride Workshop on Neuromorphic Cognition Engineering. Dan Mendat was supported by the Johns Hopkins Applied Physics Laboratory Graduate Student Fellowship. Kate Fischl was supported a National Science Foundation Graduate Research Fellowship Grant No. DGE-1232825. We also thank Prof. Jeffrey Krichmar, Tiffany Hwu and Nicolas Oros for helping us with the software of the Android based robotic platform.

Appendix C

Non-Volatile Primitives

Table C.1 *Pin-out description of the chip.* ***A/I*** *and* ***A/O*** *are analog inputs and outputs;* ***POW*** *is a power analog pad; and* ***D/I*** *and* ***D/O*** *are a digital inputs and outputs. Note that Bit-line, Source-line, Erase Gate, Coupling Gate and Word-line will be referred to with the following acronyms:* ***BL***, ***SL***, ***EG CG*** *and* ***WL***.

pin	type	name	description	val	pin	type	name	description	val
6	POW	VDD25	Supply for digital circuits	2.5v	26	POW	EG: HV-D	EG high-voltage supply	0–11.5
7	A/I	BL: B3	BL programming	0.4v	27	POW	EG: HV	EG high-voltage supply	0–11.5
14	A/I	BL: B2	BL programming	0.4v	34	POW	EG: LV+D	EG high-voltage supply	0–11.5
15	POW	GND	Global ground	0v	35	POW	EG: LV	EG high-voltage supply	0–11.5
16	A/O	BL: M76	BL Output current mirrored x76	2mA	36	POW	CG: HV-D	CG high-voltage supply	0–10.5
17	A/O	BL: M1	BL Output current mirrored x1	30μA	37	POW	CG: HV	CG high-voltage supply	0–10.5
18	A/I	WL: V2	WL control voltage	1v	38	POW	CG: LV+D	CG high-voltage supply	0–10.5
19	A/I	WL: V1	WL control voltage	1v	39	POW	CG: LV	CG high-voltage supply	0–10.5
20	D/I	CLK_IN	Clock for shift register	0/2.5v	40	D/I	AFTER	After HV-ramp signal	0/2.5v
21	D/O	D_OUT	Output data from shift register	0/2.5v	1	D/I	BEFORE	Before HV-ramp signal	0/2.5v
22	POW	SL: HV-D	SL high-voltage supply	0–4.5v	2	D/I	D_IN	Input data for shift register	0/2.5v
23	POW	SL: HV	SL high-voltage supply	0–4.5v	3	D/I	LOAD	Latch data enable	0/2.5v
24	POW	SL: LV+D	SL high-voltage supply	0–4.5v	4	A/I	REAL_VDD	Pow. supply for BL out. mirrors	0.8v
25	POW	SL: LV	SL high-voltage supply	0–4.5v	5	A/I	NM	N-mirror for programming BL	1μA

Appendix D

Dynamic Random-Access Memory Primitives (DRAM)

Table D.1 *Pin-out description of the chip.* ***A/IOs*** *are analog biases;* ***POW*** *is a power-analog signal;* ***A/O*** *is an analog output; and* ***D/I*** *and* ***D/O*** *are a digital inputs and outputs. Note that Bit-line, Word-line and Math-line will be referred to with the following acronyms:* ***BL***, ***WL***, ***ML***.

Global Port Name	Local Port Name	CID Port Name	Type	Description	Val
PLACE_PAD_i_[15:0]	port_i[15:0]	DIN[15:0]	D/I	16-bit data-input bus	0-1.2v
PLACE_PAD_i_[23:16]	port_i[23:16]	ADDR[7:0]	D/I	8-bit address bus	0-1.2v
PLACE_PAD_i_[24]	port_i[24]	BALANCE	D/I	short-circuit refresh inverter	0-1.2v
PLACE_PAD_i_[25]	port_i[25]	COMPENSATE	D/I	not($\overline{WL}$), charge inj. comp.	0-1.2v
PLACE_PAD_i_[26]	port_i[26]	$\overline{ENABLE\_OV}$	D/I	Override Broad. Switch	0-1.2v
PLACE_PAD_i_[27]	port_i[27]	PRECHARGE	D/I	Precharge ML to VDDA	0-1.2v
PLACE_PAD_i_[28]	port_i[28]	REFRESH	D/I	Write-back Refresh-enable	0-1.2v
PLACE_PAD_i_[29]	port_i[29]	WE	D/I	Write-enable for latching data	0-1.2v
PLACE_PAD_i_[30]	port_i[30]	$\overline{WL}$	D/I	Active-Low Word-line	0-1.2v
PLACE_PAD_o_[14:-2:0]	port_o[14:-2:0]	DOUT[14:-2:0]	D/O	BL_{data}[7:0] level-shifted-out	0-1.2v
PLACE_PAD_o_[15:-2:1]	port_o[15:-2:1]	DOUT[15:-2:1]	D/O	$\overline{BL}_{data}$[7:0] level-shifted-out	0-1.2v
PLACE_PAD_io_[13]	port_io[3]	OUTPUT	A/O	ML readout	0-1.2v
PLACE_PAD_io_[14]	port_io[4]	VDDA	POW	Precharge value	0.6-1.2v
PLACE_PAD_io_[16]	port_io[0]	VB_OPA_PR	A/IO	Bias for precharge-opamp	0-1.2v
PLACE_PAD_io_[17]	port_io[5]	VB_OPA_OUT	A/IO	Bias for output-opamp	0-1.2v
PLACE_PAD_io_[15]	port_io[1]	unused	–	–	–
PLACE_PAD_io_[5]	port_io[2]	unused	–	–	–

Appendix E

Suanpan: A Nano-Abacus Compute-in-Memory Processor Array

Table E.1 *NVM interface ports. S0 and S1 refer to Tables 5.1 & 5.2.*

Name	**Signal Span**	**Description**
bit1	511:0	Bit-Line control signal
bit0	511:0	Bit-Line control signal
S1_wl	127:0	Word-Line control signal
S0_wl	127:0	Word-Line control signal
S1_eg & nS1_eg	63:0	Erase-Gate control signal
S0_eg & nS0_eg	63:0	Erase-Gate control signal
S1_sl & nS1_sl	63:0	Source-line control signal
S0_sl & nS0_sl	63:0	Source-line control signal
S1_cg & nS1_cg	63:0	Coupling-Gate control signal
S0_cg & nS0_cg	63:0	Coupling-Gate control signal
nRST	511:0	Sigma Delta reset (input, negated)
nCMP	511:0	Sigma Delta comparison result (output, negated)

Table E.2 *NVM Processing-Unit (PU) biases. These biases account for both NVM row (high voltage switches) and column (ΣΔ readout) control. The following acronyms are used: bit-line (BL), word-line (WL), erase-gate (EG), source-line (SL) and coupling-gate (CG). Note: all current biases feed into a 1024/1 current mirror.*

CMP Bias name	PU Bias name	Bias type	Nom. val.	Range	Comments
BIAS_0	VDD25	Power supply	2.5v	1.75 - 2.75v	–
BIAS_16	VBC	Voltage	2v	0 - 2.5v	NVM-read Cascode
BIAS_17	VFB	P-type mirror	$205\mu A$	0-1mA (1.2v)	ΣΔ Feedback
BIAS_18	VBI	P-type mirror	$307\mu A$	0-1mA (1.2v)	ΣΔ Inverting Amp.
BIAS_19	VBF	N-type mirror	$205\mu A$	0-1mA (GND)	ΣΔ Follower
BIAS_20	VPR	Voltage	0.6/2.5v	0-2.75v	BL prog. sel/uns.
BIAS_21	VWLP	Voltage	0/1v	0-2.5v	WL prog. sel./uns.
BIAS_22	V0p5	Voltage	0.5v	0-1.2v	EG/SL uns.
BIAS_30	V4p5	Voltage	4.5v	0-4.5v	EG/SL sel.
BIAS_31	VDD_HV	Voltage	10v	0-12v	EG/CG high voltage
VSS	GND	Power Supply	0v	–	–
VDD_NET	VDD12	Power supply	1.2v	–	–

Table E.3 *Read and Write decoding*

Mode	Bits[9:6]	Target
Write	001	row mask, read and count, program and erase address, control register
	010	column mask
	011	input register
	100	accumulator parameters
Read	001	row mask, read and count, program and erase address, control register
	010	column mask
	011	input register
	100	accumulator parameters
	101	accumulator output

Table E.4 *Encoder configuration during read*

Control[4:2]	Encoder Increment	Resolution
111	1	8
110	2	7
101	4	6
100	8	5
011	16	4
010	32	3
001	64	2
000	128	1

Table E.5 *Configuration registers.*

Register name	**Description**
endcount[23:0]	Length of single accumulator cycle which control bit resolution of result
progaddr[6:0]	Address of specific row to program (0 to 127)
eraseaddr[5:0]	Address of specific row to erase (0 to 63)
ctrlreg[21:0]	Control word (described more below)
progctreg[9:0]	Length of count which determines space of control signals during program

Table E.6 *Control Register configuration.*

Mode	**Control bits**	**Function**
ERASE	ctrl_i[21]	general reset
	ctrl_i[8:2]	set ERASE length
	ctrl_i[1:0]	operation select
PROG	ctrl_i[21]	general reset
	ctrl_i[20]	one-time program
	ctrl_i[18:7]	number of program pulses
	ctrl_i[6:2]	program pulse select
	ctrl_i[1:0]	operation select
COMP	ctrl_i[21]	general reset
	ctrl_i[12]	subtraction
	ctrl_i[11:5]	mask shift value
	ctrl_i[4:2]	encoder control
	ctrl[1:0]	operation select

Accuracy	**Increment**	**End Count**
8	128	511
15	1	2^{16}-1
M	2^{16-M-1}	$2^{M+1}-1$

Table E.7 *Accumulator configuration during program*

Appendix F

Compute-in-Memory Array Processor for Artificial Intelligence Edge Computing

F.1 ESC Subsystem

Table F.1 *Processing Unit (PU) address location.*

Base Address	Peripheral	Description
0x100000	Processing Unit (ESC Subsystem)	Compute-in-Memory Unit (CIMU) containing the ESC accelerators and all supporting circuits.

Table F.2 *Memory map of the ESC Subsystem module, which is connected to the AMBA bus of a larger system. The ESC Subsystem consists of two ESC cores and a Global Interrupt Configuration address space. The peripherals listed in this table may be addressed by adding an offset of 0x100000 to the base address.*

Base Address	Peripheral	Description
0x0000	esc_core	ESC Core 0, including its peripherals
0x1000	esc_core	ESC Core 1, including its peripherals
0x4000	esc_global_conf	Global Interrupt (IRQ) Configuration

Table F.3 *Top level interface of the* ***esc_core*** *block.*

Signal Name	**Bus Width**	**Direction**	**Description**
clk	1	Input	System clock
rst_n	1	Input	Active low asynchronous reset
s0	ahb_if.slave_in (32)	Input	AMBA 3 AHB-Lite interface (trimmed)
irq2mpu_o	1	Output	IRQ status bits for MPU
irq2dma_o	4	Output	IRQ status bits for DMA
BIAS_OTA	1	Input	Pre-charge for clamping OTA
BIAS_CMP	1	Input	OTA bias
BIAS_CMP_C	1	Input	Comparator bias

F.2 Pico-Controller

Table F.4 *Memory map of the Pico Controller (**pico_conf**) peripheral which lies in the ESC Subsystem module. The configuration registers are reached by both Pico-Controller and MPU/DMA masters, allowing interaction between these modules. Absolute addresses must be calculated using Tables F.2 & F.11.*

ADDR (dec)	**ADDR (hex)**	**Byte bit**								**Dir**
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0:63	0x00:0x3F	pico_regfile[15:0][31:0]								W/R
64:65	0x40:0x41	reset_vector[15:0]								W/R
66:66	0x42:0x42							rst_soft_n	rst_hard_n	W/R
67:67	0x43:0x43									W/R
68:69	0x44:0x45	irq_config_dma[15:0]								W/R
70:71	0x46:0x47	irq_config_mpu[15:0]								W/R
72:73	0x48:0x49	irq_enable_mpu[15:0]								W/R

Table F.5 *Description of Pico-Controller memory mapped registers.*

Signal	**Description**
pico_regfile[15:0][31:0]	This is the register file for the Pico-Controller. It is 32-bits wide and contains 16 words which are addressable either locally by the Pico-Controller or globally by any master on the AMBA interconnect matrix. Note: upper 16-bits are unused.
reset_vector[15:0]	Addressable only by Masters on the AMBA interconnect matrix, this vector is the reset address of the Pico-Controller. When the Pico-Controller comes out of reset, the program counter initializes at this address.
rst_soft_n	This active-low reset is routed to the hazard unit in the Pico-Controller, and upon completion of the last instruction will force the program counter to reset state (PC=reset_vector).
rst_hard_n	This active low reset will force the Pico-Controller to reset regardless of its current state.
irq_config_dma[15:0]	This vector is used to choose which 4-bits of the *Status Register* is routed to the *irq2dma_o[3:0]* vector at the top level. This behavior is captured in Table F.10.
irq_config_mpu[15:0]	The exclusive NOR (**xnor**) between with this register and the the *Status Register* allows to select whether an interrupt is active high or active low, yielding the expression in Eq. F.1
irq_enable_mpu[15:0]	This vector masks the resulting bits from the **xnor** operation described above. The output of the mask operation undergoes a reduction operation and is wired to the *irq2mpu_o* signal, resulting in the Eq. F.2

$$irq2mpu\_premask = \overline{regfile.status \oplus irq\_config\_mpu} \tag{F.1}$$

$$irq2mpu\_o = \bigcup_i (\overline{regfile.status \oplus irq\_config\_mpu}) \cdot irq\_enable\_mpu. \tag{F.2}$$

Table F.6 *Register File of the Pico-Controller. From the perspective of the Pico-Controller, the address space described in this table should be considered* ***absolute****. Conversely, addressing the Register File from an external master requires appropriate offset calculation using using Tables F.2, F.11 & F.4. Noteworthy, all address locations may be read and written by external masters (MPU/DMA/NoC).*

ADDR (dec)	**ADDR (hex)**	**Name**	**Description**
0	0x00	Zero Register	Used in Pico-Controller as zero-register
1	0x01	Reg1	General Purpose Register used in Pico-Controller.
2	0x02	Reg2	General Purpose Register used in Pico-Controller.
3	0x03	Reg3	General Purpose Register used in Pico-Controller.
4	0x04	Reg4	General Purpose Register used in Pico-Controller.
5	0x05	Reg5	General Purpose Register used in Pico-Controller.
6	0x06	Reg6	General Purpose Register used in Pico-Controller.
7	0x07	Reg7	General Purpose Register used in Pico-Controller.
8	0x08	Reg8	General Purpose Register used in Pico-Controller.
9	0x09	Reg9	General Purpose Register used in Pico-Controller.
10	0x0A	Reg10	General Purpose Register used in Pico-Controller.
11	0x0B	Reg11	General Purpose Register used in Pico-Controller.
12	0x0C	Waveform Register	Contains all control signals for CID Analog Macro. Individual bits in this register control specific modules in the analog portion of the CID.
13	0x0D	Status Register	Status Register that allows exchange of information between Pico-Controller, ADCs, MPU and DMA. Individual bits may be polled or can also be used as maskable interrupts.
14	0x0E	Row Addr. Register	Row address for CID array, which is used when writing digital data to the analog array as well as reading data back to the digital domain via sense amplifiers. During automatic refresh, this register is updated by the Pico-Controller, however, during on demand read, any external master may update this register.
15	0x0F	Col. Addr. Register	Column address for CID array, which is used when writing digital data to the analog array as well as reading data back to the digital domain via sense amplifiers. Bits [8:6] are used to indicate the size of the transaction: [8:6] $\rightarrow$ {0: 8b, 1: 16b, 2: 32b, 3: 64b, 4: 128b, 5: 256b, 6: 512b}

Table F.7 *Pico-Controller* Waveform Register *(Addr. 0x0C of Register File in Table F.6.). All signals are active-high and interact either with digital control logic or directly with the Analog Macro in the core.*

Bit	**Signal name**	**Description**
0	wr_strobe	This signal is combined with the decoded column-address to allow data from the front-end registers to access the sense-amplifier. Data from the sense amplifier is later used to write to the CID.
1	wl_strobe	This signal is combined with the decoded row-address to allow data from the sense amplifier to written to be the CID cell.
2	equalize	Equalize DRAM sense-amplifier prior to a refresh/write operation.
3	restore	Prior to a refresh operation, this bit is used to equalize the sense amplifier by shorting its internal nodes. After the sense-amplifier is done refreshing, this bit is used to restore the data to the CID bit-cell and enable refreshed data to be read.
4	precharge	When active, the Math-Line (ML) is clamped using a unit feedback OTA, resulting in VML=VPR+offset. When this signal is disabled, the Math-Line is clamped to the supply VPR, resulting in VML=VPR.
5	start_adc	This signal is used to initiate and analog-to-digital conversion in either the SAR or SSL ADCs.
6	compute	This signal is used to initiate and analog-to-digital conversion in either the SAR or SSL ADCs.
7	rst_hc	Reset for adiabatic charge recycling computing mode. During a compute phase, all columns maybe held at ground with this signal, and upong release, charge from and external coil can be used to compute in the CID array.
8	rd_strobe_sense	This signal is used to write refreshed data from the sense amplifier to the interface registers, which in turn are memory mapped.
9	en_ota_bias	This enabled signal is combined with memory mapped registers to provide row-wise bias-gating of the Operational Transconductance Amplifiers (OTA) that are present in the readout circuitry.
10	en_cmp_bias	This enabled signal is combined with memory mapped registers to provide row-wise bias-gating for the comparators that are part of the single-slope ADC.
11	Not Used	
12	Not Used	
13	Not Used	
14	Not Used	
15	Not Used	

Table F.8 Waveform Register *templates for different CID array operations. The bottom-most row shows the timing sequence for each operation. During each time-step (t_i), the values of the* Waveform Register *may be held for an arbitrary number of clock cycles from within the Pico-Controller. The second-to-bottom row shows the 11-bit value converted to decimal.*

Bit	**Signal Name**	**Operation**																			
		RST	**WR**					**RD**						**Compute**							
0	**wr_strobe**	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	**wl_strobe**	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0
2	**equalize**	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	**restore**	1	1	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1
4	**precharge**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1
5	**start_adc**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
6	**compute**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
7	**rst_hc**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	**rd_strobe_sense**	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
9	**en_ota_bias**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
10	**en_cmp_bias**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0
Decimal Value		24	28	16	19	18	26	28	16	18	282	26	24	24	1560	1544	1640	1608	1544	1560	24
Time-step		t_0	$t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4$					$t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5$						$t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \rightarrow t_6 \rightarrow t_7$							

Table F.9 *Pico-Controller* Status Register *(Addr. 0x0D of Register File in Table F.6).*

Bit	**Signal name**	**Description**
0	status_bit_0	General purpose status bit.
1	status_bit_1	General purpose status bit.
2	status_bit_2	General purpose status bit.
3	status_bit_3	General purpose status bit.
4	status_bit_4	General purpose status bit.
5	status_bit_5	General purpose status bit.
6	status_bit_6	General purpose status bit.
7	status_bit_7	General purpose status bit.
8	status_bit_8	General purpose status bit.
9	status_bit_9	General purpose status bit.
10	status_bit_10	General purpose status bit.
11	status_bit_11	General purpose status bit.
12	ssl_working	Indicates that the single slope ADC is working on a conversion.
13	ssl_done	Indicates that the single slope ADC has finished a conversion. This bit will stay high until a new conversion starts.
14	ssl_busy	Indicates that the SAR-ADC is working on a conversion.
15	sar_data_avail	Indicates that the SAR-ADC has finished a conversion. This bit will stay high until all data is read from the buffer.

F.3 ESC Core

Table F.10 *Wiring between* Status Register *and DMA IRQs. Status Register description may be found in Table F.9. In this table,* regfile.status *refers to the status register.*

IRQ Vector	Selection Bits
irq2dma_o[0]	⇐ regfile.status[irq_config_dma[0:3]
irq2dma_o[1]	⇐ regfile.status[irq_config_dma[4:7]
irq2dma_o[2]	⇐ regfile.status[irq_config_dma[8:11]
irq2dma_o[3]	⇐ regfile.status[irq_config_dma[12:15]

Table F.11 *Memory map of the ESC Core (**esc_core**) peripheral which lies in the ESC Subsystem module. Absolute addresses may be calculated using Table F.2.*

Base Address	End Address	Peripheral	Description
0x0000	0x03FF	pico_mem	Pico-Controller Instruction Memory, which consists of a 32×256 address space. Note: The upper 6-bits of the 32-bit word are not used.
0x0400	0x05FF	CID_FE	Input/Output and Polarity Registers for CID
0x0600	0x07FF	ADC_SS	Single Slope and SAR ADC Configuration Registers
0x0800	0x087F	pico_conf	Pico- Controller Configuration Registers

Table F.12 *Memory map of the CID Front-End Interface (**CID_FE**) peripheral, which lies in the ESC Core module. Absolute addresses may be calculated using Table F.2.*

ADDR (dec)	ADDR (hex)	Byte bit								Dir
		7	6	5	4	3	2	1	0	
0:63	0x00:0x3F	cid_data_bl/q_sense[31:0][15:0]								W/R
64:127	0x40:0x7F	cid_polarity[31:0][15:0]								W/R
128:191	0x80:0xBF	cid_data_x[31:0][15:0]								W/R

Table F.13 *Register description of the CID Front-End peripheral memory space.*

Signal	**Description**
cid_data_bl[31:0][15:0]	These 512 registers are split into 16 addressable words by Byte. These registers feed data to the CID Bit-Lines, which are used to store parameters (w_i) in the CID array.
q_sense[31:0][15:0]	These 512 registers represent read data (w_i) that originates in the sense amplifiers in the refresh circuitry. These registers conveniently share the same address space as *cid_data_bl[31:0][15:0].*
cid_polarity[31:0][15:0]	These 512 registers are set as an experimental sign-extension mode for each *x-input* during computation. The default value is 0, which is used to five a positive sign to x_i.
cid_data_x[31:0][15:0]	These 512 registers are used as *x-inputs* for computing in the CID array.

F.4 SSL ADC Peripheral

Table F.14 *Memory map of Successive Approximation Register (SAR) and Single Slope (SSL) Analog to Digital Converters (**ADC_SS**). Absolute addresses must be calculated using Tables F.2 & F.11.*

ADDR (hex)	**ADDR (dec)**	**Byte bit**								**Dir**
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0x000:0x07F	0:127	sl_rdata[7:0]∗128								R
0x080:0x080	128:128	sar_rdata[7:0]								R
0x081:0x0FF	129:255									
0x100:0x10F	256:271	en_row4comp[127:0]								W/R
0x110:0x111	272:273	sar_clk_div[9:0]								W/R
0x112:0x112	274:274	sar_settle_t		sar_precision			sar_clk_en	sar_chip_sel	sar_rst_ctrl_n	W/R
0x113:0x113	275:275							sar_data_avail	sar_busy	W/R
0x114:0x114	276:276	sl_precharge_time[7:0]								W/R
0x115:0x115	277:277	sl_tick_ramp_dig[7:0]								W/R
0x116:0x116	278:278	sl_tick_ramp_ana[7:0]								W/R
0x117:0x117	279:279									
0x118:0x119	280:281	sl_ramp_dig_ini[15:0]								W/R
0x11A:0x127	282:295	sl_grid_dig[15:0]∗7								W/R
0x128:0x137	296:311	sl_ramp_dig_step[15:0]∗8								W/R
0x138:0x139	312:313	sl_counter_end_dig[15:0]								W/R
0x13A:0x13B	314:315	sl_init_addr_ana[8:0]								W/R
0x13C:0x13D	316:317	sl_step_addr_ana[8:0]								W/R
0x13E:0x13F	318:319	sl_init_x_ana[9:0]								W/R
0x140:0x140	320:320	sl_step_x_ana[5:0]								W/R
0x141:0x141	321:321						ssl_sel_ramp_bits[2:0]			W/R
0x142:0x142	322:322		sl_done	sl_working	sl_full_fifo		en_row4ramp[128]	sl_rst_fifo_n	ssl_rst_ctrl_n	W/R
0x143:0x143	323:323									
0x144:0x153	324:339	sl_fifo_empty_flag_i[127:0]								W/R

Table F.15 *Single Slope ADC (SSL) configuration.*

Signal	Description
ssl_rdata[7:0]∗128	128 Byte SSL ADC read data.
sar_rdata[7:0]	1 Byte SAR ADC read data.
en_row4comp[127:0]	128-bits used to enable or disable any of the 128 rows for computing. These bits are combined with bias enables from the Pico-Controller to provide fine control over row-wise bias gating.
sar_clk_div[9:0]	Clock divider value for SAR ADC.
sar_settle_t[1:0]	Settle time for SAR ADC. This parameter allows up to 4 SAR clock cycles for the voltage to settle on the SAR capacitor during the sample phase.
sar_precision[3:0]	The SAR ADC may operate with precision on demand, allowing for shorter conversion times with lower output precision. This parameters allows anything between 1-bit and 8-bit precision on the output. Noteworthy, there is a constant overhead of approx. 4 ADC clock cycled for any conversion.
sar_clk_en	Clock enable for the SAR ADC.
sar_chip_sel	Chip Select for the SAR ADC allows disabling the output when more than one SAR is used in parallel. The ESC Subsystem only has 1 SAR ADC.
sar_rst_ctrl_n	Active low synchronous reset for the SAR ADC.
sar_data_avail	Indicates that data is available in the output buffer. This bit is also mapped to the register file in the Pico-Controller, thus allowing it to be polled or to be used as an interrupt.
sar_busy	Indicates that SAR is busy. This bit is also mapped to the register file in the Pico-Controller, thus allowing it to be polled or to be used as an interrupt.
ssl_precharge_time[7:0]	SSL ADC precharge time allows OTA outputs to reach a stable value before initializing the analog/digital ramp.
ssl_tick_ramp_dig[7:0]	Clock divider value for the digital ramp. This value determines the effective tick period.
ssl_tick_ramp_ana[7:0]	Clock divider value for the analog ramp generation. This value is used to program the time interval for loading the front-end registers that control the analog ramp value.
ssl_ramp_dig_ini[15:0]	Initial value for the digital ramp.

Table continues on next page.

Table F.15 *Single Slope ADC (SSL) configuration (continued).*

Signal	Description
ssl_grid_dig[15:0]∗7	This set of parameters is used to program the domain value of the vertexes at a Piece Wise Linear (PWL) function changes its first derivative (step). The PWL function has 8 points, but since the first value is always zero, there no no need to include it.
ssl_ramp_dig_step[15:0]∗8	This set of parameters represents the first derivative value between the vertexes defined in the previous item.
ssl_counter_end_dig[15:0]	This is the saturation value for the digital counter (domain for digital grid).
ssl_init_addr_ana[8:0]	Address at which the SSL DAC starts loading logic ones (address range $\in [0:511]$).
ssl_step_addr_ana[8:0]	Address step size for SSL DAC. This is the effective increment at which a spatial PWM of logic ones is loaded into the DAC front-end registers.
ssl_init_x_ana[9:0]	Initial value for DAC front-end registers.
ssl_step_x_ana[5:0]	Spatial PWM that is loaded sequentially into the front-end registers of the DAC during analog ramp generation. The maximum allowed value is 32.
ssl_sel_ramp_bits[2:0]	This parameter allows the user to select which 8 bits from the 16 bit digital ramp will be stored in the ouput of the ADC.
ssl_done	Indicates that the SSL ADC has completed a conversion. This bit is also mapped to the register file in the Pico-Controller, thus allowing it to be polled or to be used as an interrupt.
ssl_working	Indicates that SSL ADC is busy. This bit is also mapped to the register file in the Pico-Controller, thus allowing it to be polled or to be used as an interrupt.
ssl_full_fifo	This bit represents the logic OR between the full flags of all 128 output fifos. This bit does take into consideration that some rows may be disabled. The *ssl_full_fifo* flag will go low once all data has been read from the output buffers.
en_row4ramp[128]	This parameter allows the user to disable the analog ramp. This is done by gating the bias of the OTA.
ssl_rst_fifo_n	Active low reset of ADC FIFO's that are at the output of every row.

Table continues on next page.

Table F.15 *Single Slope ADC (SSL) configuration (continued).*

Signal	**Description**
ssl_rst_ctrl_n	Active low reset of the SSL finite state machine controller.
ssl_fifo_empty_flag_i[127:0]	Memory mapped empty flags for the SSL ADC output fifos.

F.5 ESC Analog

Table F.16 *Top level interface of the* ***esc_analog*** *block.*

Signal Name	**Bus Width**	**Direction**	**Description**
x_ramp_i	512	Input	*x-inputs* that are shifted in for DAC ramp.
x_cid_i	512	Input	*x-inputs* for CID computing.
col_sel_n_i	512	Input	Column select for CID writing.
bl_cid_i	512	Input	Input Data for writing to the CID.
sa_equalize_i	1	Input	Equalize for the sense amplifier (refresh-read-write).
sa_restore_i	1	Input	Restore refreshed value back to the CID array.
wr_strobe_n_i	1	Input	Write-strobe for writing new data, overrides sense amplifier.
rst_hc_i	1	Input	Reset for charge-recycling computation circuitry.
en_ota_bias_i	129	Input	Enables clamp OTA attached to ML[128] that goes to comparator.
en_cmp_bias_i	128	Input	Enables comparator biases.
en_cmp_output_i	1	Input	Enable the output of the SSL comparator.
wordline_i	129	Input	129 word-lines (wl). *wl[128]* is used for ramp-cid.
ota_prech_i	1	Input	Pre-charge for clamping OTA.
BIAS_OTA	1	Input	OTA bias.
BIAS_CMP	1	Input	Comparator bias.
BIAS_CMP_C	1	Input	Comparator bias (cascode).
sa_o	512	Output	Output of the sense amplifiers obtained after a refresh cycle.
ota_o	128	Output	128 analog outputs that may (or may not) go to SAR ADCs.
cmp_o	128	Output	Comparison between the analog ramp and CID compute result.

References

[1] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[2] B. Hoeneisen and C. A. Mead, "Fundamental limitations in microelectronics—i. mos technology," *Solid-State Electronics*, vol. 15, no. 7, pp. 819–829, 1972.

[3] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[4] D. W. Tank and J. Hopfield, "Neural computation by concentrating information in time," *Proceedings of the National Academy of Sciences*, vol. 84, no. 7, pp. 1896–1900, 1987.

[5] G. E. Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[6] "Summary of TSMC's Major Future R&D Projects ," https://www.tsmc.com/english/dedicatedFoundry/technology/future_rd.htm, Accessed: October 04, 2020.

[7] G. T. Tuttle, S. Fallahi, and A. A. Abidi, "A low-power analog CMOS vector quantizer," in *Proceedings 1993 Data Compression Conference (DCC '93)*, 1993, pp. 410–419.

[8] K. Yang and A. G. Andreou, "Multiple input floating-gate MOS differential amplifiers and applications for analog computation," in *36th Midwest Symposium on Circuits and Systems (MWSCAS)*, 1993, pp. 1212–1216.

[9] ——, "A multiple input differential amplifier based on charge sharing on a floating-gate MOSFET," *Analog Integrated Circuits and Signal Processing*, vol. 6, pp. 197–208, 1994.

[10] S. Nakamura and Y. Nagazumi, "A matched filter design by charge-domain operations," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 52, no. 5, pp. 867–874, May 2005.

[11] B. Sadhu, M. Sturm, B. M. Sadler, and R. Harjani, "Analysis and Design of a 5 GS/s Analog Charge-Domain FFT for an SDR Front-End in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 5, pp. 1199–1211, May 2013.

[12] A. J. Agranat, C. F. Neugebauer, R. D. Nelson, and A. Yariv, "The CCD neural processor: A neural network integrated circuit with 65536 programmable analog synapses," *IEEE Transactions on Circuits and Systems*, vol. 37, no. 8, pp. 1073–1075, 1990.

[13] Z. Tang, O. Ishizuka, and H. Matsumoto, "Programmable MOS Charge-Mode Neural Circuits," *IET Electronics Letters*, vol. 28, no. 22, pp. 2059–2060, 1992.

[14] R. Genov and G. Cauwenberghs, "Charge-mode parallel architecture for vector-matrix multiplication," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 10, pp. 930–936, Oct. 2001.

[15] R. Karakiewicz, R. Genov, and G. Cauwenberghs, "1.1 TMACS/mW Fine-Grained Stochastic Resonant Charge-Recycling Array Processor," *IEEE Sensors Journal*, vol. 12, no. 4, pp. 785–792, 2012.

[16] G. Tognetti, J. Sengupta, P. O. Pouliquen, and A. G. Andreou, "Characterization of a pseudo-DRAM Crossbar Computational Memory Array in 55nm CMOS," in *53rd Annual Conference on Information Sciences and Systems (CISS)*, Mar. 2019, pp. 1–5.

[17] M. Noack, C. Mayr, J. Partzsch, M. Schultz, and R. Schueffny, "A switched-capacitor implementation of short-term synaptic dynamics," in *19th IEEE International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES 2012)*, 2012, pp. 214–218.

[18] C. Mayr, J. Partzsch, M. Noack, S. Hänzsche, S. Scholze, S. Höppner, G. Ellguth, and R. Schueffny, "A Biological-Realtime Neuromorphic System in 28 nm CMOS using Low-Leakage Switched Capacitor Circuits," *arXiv.org*, p. arXiv:1412.3233, Dec. 2014.

[19] M. Noack, M. Krause, C. Mayr, J. Partzsch, and R. Schueffny, "VLSI Implementation of a Conductance-based Multi-Synapse using Switched-Capacitor Circuits," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Feb. 2014, pp. 1–4.

[20] D. Bankman and B. Murmann, "An 8-bit, 16 input, 3.2 pJ/op switched-capacitor dot product circuit in 28-nm FDSOI CMOS," in *2016 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, Nov. 2016, pp. 21–24.

[21] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-Tile 2.4-Mb In-Memory-Computing CNN Accelerator Employing Charge-Domain Compute," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019.

[22] K. A. Sanni, "Heterogeneous Chip Multiprocessor: Data Representation, Mixed-Signal Processing Tiles, and System Design," Ph.D. dissertation, Ph.D. Dissertation, Johns Hopkins University, 2019.

[23] K. Sanni, T. Figliolia, G. Tognetti, P. O. Pouliquen, and A. G. Andreou, "A Charge-Based Architecture for Energy-Efficient Vector-Vector Multiplication in 65nm CMOS," *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2018.

[24] K. Thurber and L. Wald, "Associative and Parallel Processors," *Computing Surveys*, vol. 7, no. 4, Dec. 1975.

[25] J. D. Roberts, "Associative processing: a paradigm for massively parallel AI," in *AAAI Technical Report SS-93-04*, 1993, pp. 187–192.

[26] S. Hedge, I. Jalowiecki, and R. Lea, "WASP 3: A real time signal and data processor," in *Proceedings 4th International Conference on Wafer Scale Integration*, Dec. 1991, pp. 105–114.

[27] R. Lea, "VLSI and WSI associative string processors for structured data processing," *IEE Proceedings Part-I: Communications, Speech and Vision*, vol. 133, no. 3, pp. 113–122, Jun. 1986.

[28] F. Herrmann and C. G. Sodini, "A 256-element associative parallel processor," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 4, pp. 365–370, 1995.

[29] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the Terasys massively parallel PIM array," *IEEE Computer*, vol. 28, no. 4, pp. 23–31, Apr. 1995.

[30] D. A. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, 1997.

[31] E. Upchurch, T. L. Sterling, and J. Brockman, "Analysis and modeling of advanced PIM architecture design tradeoffs," in *Innovative Architecture for Future Generation High-Performance Processors and Systems, 2003*, 2003, pp. 66–75.

[32] M. Beretta, A. Annovi, A. Andreani, M. Citterio, A. Colombo, V. Liberali, S. Shojaii, A. Stabile, R. Beccherle, P. Giannetti, and F. Crescioli, "Next

generation associative memory devices for the FTK tracking processor of the ATLAS experiment," *Journal of Instrumentation*, vol. 9, no. 03, pp. C03 053–C03 053, Mar. 2014.

[33] J. Lubkin and G. Cauwenberghs, "VLSI implementation of fuzzy adaptive resonance and learning vector quantization," *Analog Integrated Circuits and Signal Processing*, vol. 30, no. 2, pp. 149–157, 2002.

[34] T. Serrano-Gotarredona and B. Linares-Barranco, "A Modified ART 1 Algorithm more suitable for VLSI Implementations," *Neural Networks*, vol. 9, no. 6, pp. 1025–1043, 1996.

[35] M. H. Cohen, P. A. Abshire, and G. Cauwenberghs, "Mixed-mode VLSI implementation of fuzzy ART," in *1998 IEEE International Symposium on Circuits and Systems (ISCAS)*, 1998, pp. 251–254.

[36] P. O. Pouliquen, A. G. Andreou, and K. Strohben, "Winner-Takes-All Associative Memory: A Hamming Distance Vector Quantizer," *Analog Integrated Circuits and Signal Processing*, vol. 13, no. 1-2, May 1997.

[37] K. A. Boahen, P. O. Pouliquen, A. G. Andreou, and R. E. Jenkins, "A heteroassociative memory using current-mode MOS analog VLSI circuits," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 5, pp. 747–755, 1989.

[38] M. A. Marwick and A. G. Andreou, "Retinomorphic system design in three dimensional soi-cmos," in *2006 IEEE International Symposium on Circuits and Systems*. IEEE, 2006, pp. 4–pp.

[39] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[40] B. R. Gaines, "Stochastic Computing Systems," in *Advances in Information Systems Science*, J. F. Tou, Ed. New York: Plenum, 1969, pp. 38–172.

[41] W. Poppelbaum, "Statistical processors," in *Advances in Computers*. Elsevier, 1976, vol. 14, pp. 187–230.

[42] S. T. Ribeiro, "Random-pulse machines," *IEEE Transactions on Electronic Computers*, no. 3, pp. 261–276, 1967.

[43] A. Alaghi and J. P. Hayes, "Survey of Stochastic Computing," *ACM Transactions on Embedded Computing Systems (TECS)*, pp. 1–25, 2012.

[44] P. K. Gupta and R. Kumaresan, "Binary multiplication with pn sequences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 603–606, 1988.

[45] E. H. Dinan and B. Jabbari, "Spreading codes for direct sequence cdma and wideband cdma cellular networks," *IEEE communications magazine*, vol. 36, no. 9, pp. 48–54, 1998.

[46] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. USA: Omnipress, 2011, pp. 921–928. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104482.3104598

[47] R. Kasturi, D. B. Goldgof, R. Ekambaram, G. Pratt, E. Krotkov, D. D. Hackett, Y. Ran, Q. Zheng, R. Sharma, M. Anderson, M. Peot, M. Aguilar, D. Khosla, Y. Chen, K. Kim, L. Elazary, R. C. Voorhies, D. F. Parks, and L. Itti, "Performance evaluation of neuromorphic-vision object recognition algorithms," in *2014 22nd International Conference on Pattern Recognition*, Aug 2014, pp. 2401–2406.

[48] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[49] A. S. Cassidy, R. Alvarez-Icaza, F. Akopyan, J. Sawada, J. V. Arthur, P. A. Merolla, P. Datta, M. G. Tallada, B. Taba, A. Andreopoulos, A. Amir, S. K. Esser, J. Kusnitz, R. Appuswamy, C. Haymes, B. Brezzo, R. Moussalli, R. Bellofatto, C. Baks, M. Mastro, K. Schleupen, C. E. Cox, K. Inoue, S. Millman, N. Imam, E. McQuinn, Y. Y. Nakamura, I. Vo, C. Guo, D. Nguyen, S. Lekuch, S. Asaad, D. Friedman, B. L. Jackson, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "Real-time scalable cortical computing at 46 giga-synaptic OPS/watt with 100× speedup in time-to-solution and 100,000× reduction in energy-to-solution," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. IEEE Press, Nov. 2014.

[50] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, E. McQuinn, B. Shaw, N. Pass, and D. S. Modha, "Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores," in *Proceedings of the 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–10.

[51] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE Computer Society, Nov. 2012, pp. 1–11.

[52] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[53] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "QUEST: A 7.49TOPS multi-purpose log-quantized DNN inference engine stacked on 96MB 3D SRAM using inductive-coupling technology in 40nm CMOS," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*. IEEE, Mar. 2018, pp. 216–218.

[54] J. Zhang, Z. Wang, and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," *IEEE Journal of Solid-State Circuits*, pp. 1–10, 2017.

[55] M. Gao, C. Delimitrou, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and C. Kozyrakis, "DRAF - A Low-Power DRAM-Based Reconfigurable Acceleration Fabric." *IEEE Micro*, vol. 37, no. 3, pp. 70–78, 2017.

[56] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," *arXiv.org*, p. arXiv:1611.09988, Nov. 2016.

[57] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2019.

[58] "The Cerebras Wafer Scale Engine," https://www.cerebras.net/product/#chip, Accessed: October 05, 2020.

[59] "Cloud Tensor Processing Units (TPUs)," https://cloud.google.com/tpu/docs/tpus, Accessed: October 05, 2020.

[60] Department of Defense Fiscal Year (FY) 2018 Budget Estimates, *Defense-Wide Justification Book Volume 1 of 1*, 2017, Accessed: September 22, 2020. [Online]. Available: https://www.darpa.mil/attachments/1TheCaseforSecureASICs_Slides.pdf

[61] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, 2003.

[62] J. Sengupta, G. Tognetti, P. O. Pouliquen, and A. G. Andreou, "Multilevel storage cell characterization and behavior modeling of a crossbar computational array in esf3 flash technology : (invited paper)," in *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*, 2019, pp. 1–5.

[63] N. Do, "Scaling of split-gate flash memory and its adoption in modern embedded non-volatile applications," in *2016 International Conference on IC Design and Technology (ICICDT)*, 2016, pp. 1–4.

[64] N. Do, L. Tee, S. Hariharan, S. Lemke, M. Tadayoni, W. Yang, M. Wu, J. Kim, Y. Chen, C. Su, V. Tiwari, S. Zhou, R. Qian, and I. Yue, "A 55 nm logic-process-compatible, split-gate flash memory array fully demonstrated at automotive temperature with high access speed and reliability," in *2015 IEEE International Memory Workshop (IMW)*, 2015, pp. 1–3.

[65] S. Sze, *Physics of Semiconductor Devices*. John Wiley & Sons, Nov. 1981.

[66] J. M Rabaey, A. P Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 2003.

[67] R. E. Matick and S. E. Schuster, "Logic-based edram: Origins and rationale for use," *IBM Journal of Research and Development*, vol. 49, no. 1, pp. 145–165, 2005.

[68] G. Wang, D. Anand, N. Butt, A. Cestero, M. Chudzik, J. Ervin, S. Fang, G. Freeman, H. Ho, B. Khan, B. Kim, W. Kong, R. Krishnan, S. Krishnan, O. Kwon, J. Liu, K. McStay, E. Nelson, K. Nummy, P. Parries, J. Sim, R. Takalkar, A. Tessier, R. M. Todi, R. Malik, S. Stiffler, and S. S. Iyer, "Scaling deep trench based edram on soi to 32nm and beyond," in *2009 IEEE International Electron Devices Meeting (IEDM)*, 2009, pp. 1–4.

[69] R. Giterman, A. Shalom, A. Burg, A. Fish, and A. Teman, "A 1-mbit fully logic-compatible 3t gain-cell embedded dram in 16-nm finfet," *IEEE Solid-State Circuits Letters*, vol. 3, pp. 110–113, 2020.

[70] H. Pilo, D. Anand, J. Barth, S. Burns, P. Corson, J. Covino, and S. Lamphier, "A 5.6-ns random cycle 144-mb dram with 1.4 gb/s/pin and ddr3-sram interface," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 11, pp. 1974–1980, 2003.

[71] W. S. Boyle and G. E. Smith, "Charge coupled semiconductor devices," *The Bell System Technical Journal*, vol. 49, no. 4, pp. 587–593, April 1970.

[72] Parrish and McVey, "Implications of charge-coupled devices for pattern recognition," *IEEE Transactions on Computers*, vol. C-25, no. 11, pp. 1146–1152, Nov 1976.

[73] R. Karakiewicz, R. Genov, and G. Cauwenberghs, "480-GMACS/mW Resonant Adiabatic Mixed-Signal Processor Array for Charge-Based Pattern Recognition," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 11, pp. 2573–2584, 2007.

[74] ——, "1.1 tmacs/mw fine-grained stochastic resonant charge-recycling array processor," *IEEE Sensors Journal*, vol. 12, no. 4, pp. 785–792, 2011.

[75] B. Keeth and R. J. Baker, *DRAM circuit design: a tutorial*. Wiley-IEEE Press, 2000.

[76] R. Sarpeshkar, T. Delbruck, and C. A. Mead, "White noise in mos transistors and resistors," *IEEE Circuits and Devices Magazine*, vol. 9, no. 6, pp. 23–29, Nov. 1993.

[77] G. Tognetti, J. Sengupta, P. O. Pouliquen, and A. G. Andreou, "Characterization of a pseudo-dram crossbar computational memory array in 55nm cmos : (invited paper)," in *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*, 2019, pp. 1–5.

[78] E. Quinnell, E. E. Swartzlander, and C. Lemonds, "Floating-point fused multiply-add architectures," in *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*. IEEE, 2007, pp. 331–337.

[79] E. R. Hsieh, C. H. Chuang, and S. S. Chung, "An innovative 1t1r dipole dynamic random access memory (diram) featuring high speed, ultra-low power, and low voltage operation," in *2016 International Symposium on VLSI Technology, Systems and Application (VLSI-TSA)*, April 2016, pp. 1–2.

[80] W. Bux, F. Closs, K. Kuemmerle, H. Keller, and H. Mueller, "Architecture and design of a reliable token-ring network," *IEEE Journal on Selected Areas in Communications*, vol. 1, no. 5, pp. 756–765, November 1983.

[81] *AMBA 3 AHB-Lite Protocol*, ARM, 2006, v1.0.

[82] R. Schreier, G. C. Temes *et al.*, *Understanding delta-sigma data converters*. IEEE press Piscataway, NJ, 2005, vol. 74.

[83] E. Culurciello, R. Etienne-Cummings, and K. A. Boahen, "A biomorphic digital image sensor," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 2, pp. 281–294, 2003.

[84] Y. Moon and D.-K. Jeong, "An efficient charge recovery logic circuit," *IEICE transactions on electronics*, vol. 79, no. 7, pp. 925–933, 1996.

[85] R. Karakiewicz, R. Genov, and G. Cauwenberghs, "1.1 tmacs/mw load-balanced resonant charge-recycling array processor," in *2007 IEEE Custom Integrated Circuits Conference*. IEEE, 2007, pp. 603–606.

[86] C. Lynch, "How do your data grow?" *Nature*, vol. 455, no. 7209, pp. 28–29, 2008.

[87] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[88] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[89] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[90] "Arduino Products," https://www.arduino.cc/en/Main/Products, Accessed: October 25, 2020.

[91] "Freedom Everywhere: HiFive1," https://www.sifive.com/boards/hifive1, Accessed: October 25, 2020.

[92] "RISCV Core IP: S7 Series."

[93] C. A. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990.

[94] A. S. Cassidy, J. Georgiou, and A. G. Andreou, "Design of silicon brains in the nano-CMOS era: spiking neurons, learning synapses and neural architecture optimization," *Neural Networks*, vol. 45, pp. 4–26, Jun. 2013.

[95] F. C. Morabito, A. G. Andreou, and E. Chicca, "Neuromorphic engineering: from neural systems to brain-like engineered systems," *Neural Networks*, vol. 45, pp. 1–3, May 2013.

[96] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, 2015.

[97] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing," *arXiv.org*, Mar. 2016.

[98] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, 1st ed. The MIT Press, Jul. 2009.

[99] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[100] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *arXiv.org*, Apr. 2016.

[101] D. A. Pomerleau, "ALVINN, an autonomous land vehicle in a neural network," Tech. Rep. CMU-CS-89-107, Jan. 1989.

[102] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-Road Obstacle Avoidance Through End-to-End Learning," in *Advances in Neural Information Processing Systems 18 (NIPS-2005)*, 2005.

[103] T. Hwu, J. Isbell, N. Oros, and J. Krichmar, "A Self-Driving Robot Using Deep Convolutional Neural Networks on Neuromorphic Hardware," *arXiv.org*, Nov. 2016.

[104] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2011.

Vita

Gaspar Tognetti was born in 1984 in San Carlos de Bariloche, Argentina. He received a degree in Electronics Engineering from Universidad Nacional del Sur, Bahía Blanca in 2012. In 2013, he started his PhD at the Sensory Communications and Microsystems Lab with Prof. Andreas Andreou at the Johns Hopkins University. He received a Master in Science of Engineering (MSE) degree in 2015 from the same institution. His research focuses on bio-inspired microsystems for hardware AI and Machine Intelligence.